UNIVERSITÀ DELLA CALABRIA

Dipartimento di Ingegneria Informatica,
Modellistica, Elettronica e Sistemistica

Corso di Laurea Specialistica in

Ingegneria Informatica

Indirizzo Reti

Tesi di Laurea

Un'Infrastruttura di Sviluppo Web Enterprise Distribuita su Cloud Basata su Modelli PaaS e IaaS

Relatori

Prof. Domenico Talia
Ing. Fabrizio Scarcello

Candidato

Natale Vinto
Matr. 137523

Anno Accademico 2012/2013

Ai miei genitori, grazie

Indice generale

INTRODUZIONE

A post-industrial society is based on services.

La società dei servizi, per Daniel Bell, è la società dell'informazione che produce economia della conoscenza, a valle di uno sviluppo industriale la cui curva di evoluzione tende asintoticamente ad una soglia fisica di produzione, ormai prossima.
Ciò che maggiormente interpreta il senso di queste considerazioni, è il paradigma del cloud computing, un modello di computazione on-demand, attraverso il quale vengono erogati dei servizi che permettono di accedere a delle risorse distribuite in maniera semplice, immediata e del tutto trasparente da parte di chi le utilizza.
Il modello cloud è in continua evoluzione. Sebbene si cerchi di convergere a livello concettuale su un'unica definizione, lo stesso termine cloud è rimasto per molto tempo strettamente legato a determinati provider e a delle stringenti soluzioni offerte dai medesimi nel mercato IT. L'obiettivo di questo elaborato è quello di descriverne lo stato dell'arte e di proporre un'infrastruttura di sviluppo Web Enterprise per cloud computing, aperta, estendibile, basata su modelli di servizio ma, soprattutto, compatibile fra i cloud.

Lo sviluppo Web è l'aspetto più comune ad aziende ed enti che vogliono essere visibili sul mercato; avere la possibilità di poter erogare dei servizi in questo senso, attraverso un sistema altamente scalabile e largamente distribuito quale è il cloud, fornisce delle opportunità, che fino a qualche anno fa erano prerogativa soltanto di pochi attori, perlopiù grosse Corporation capaci di possedere ingenti quantità di hardware.

Nel corso di questo lavoro di tesi, viene fornita una descrizione teorica e formale sul cloud computing, espresso come modello di erogazione di servizi, che a sua volta, al suo interno, comprende dei modelli di servizio e di deployment. Nel corso della discussione nel primo capitolo, il paradigma del cloud viene costantemente paragonato alle griglie computazionali, per fornire un nesso storico e logico sul perché di alcune caratteristiche e di talune soluzioni di problemi comuni alla computazione distribuita e on-demand. Il cloud utilizza il grid computing come dorsale di comunicazione

all'interno di una totalità di modelli di computazionale preesistenti, verso un'architettura a servizi erogati attraverso il Web. Il capitolo prosegue con la descrizione dell'architettura del cloud, analizzando i componenti fondamentali del modello computazionale e del modello dati attraverso la virtualizzazione delle risorse e l'erogazione delle stesse come servizi. Viene inoltre posta una maggiore enfasi sul sistemi di virtualizzazioni esistenti, descrivendone caratteristiche e funzionalità, e viene infine discusso l'aspetto della sicurezza in una struttura largamente condivisa e complessa come quella interpretata dal paradigma in esame.

Nel secondo capitolo viene fornita una vasta ed esaustiva panoramica sui modelli di servizio IaaS e PaaS attualmente presenti sul mercato ed in letteratura, al fine perseguire gli scopi prefissati precedentemente, ovvero ricercare tutte le possibili combinazioni dei cloud per ottenere una piattaforma di sviluppo web aperta ed estendibile, su cloud computing. Vengono discusse in maniera dettagliata sia l'intera offerta IaaS con soluzioni quali Amazon, OpenNebula, Joyent, Openstack, sia le maggiori PaaS quali App Engine, Heroku, CloudFoundry, Openshift. Nella descrizione delle varie soluzioni, si è posta maggiore enfasi su quelle di natura open source, target naturale di una ricerca di questo tipo, verso l'interoperabilità dei cloud. A tal fine, il capitolo si conclude con la descrizione di DeltaCloud, un progetto il cui scopo prefissato è quello di fornire un alto livello di astrazione nel cloud, attraverso delle API che permettono la portabilità delle stesse applicazioni sulle maggiori soluzioni di infrastruttura.

Il terzo capitolo comprende una discussione dettagliata sulle soluzioni scelte nella fase implementativa per realizzare il “super-modello” PaaS over IaaS, ovvero Openstack ed Openshift le quali, per la loro natura open source e per la loro forte presa nella comunità di sviluppatori, stanno diventando lo standard de facto di soluzioni open source, rispettivamente di infrastruttura e di piattaforma. Viene fornita una descrizione dettagliata su Openstack, sulle unità che lo compongono e sui modelli di deployment ottenibili mediante il suo utilizzo. Viene inoltre posta una particolare attenzione sui servizi di orchestrazione, Heat e Juju, ed in particolare sul primo, il quale rappresenta la scelta di default dell'infrastruttura. L'orchestrazione delle risorse è un aspetto cruciale nel cloud, ed Openstack espone tramite Heat dei servizi, alla stregua di tutti gli altri, per la descrizione delle risorse e di definizione delle metriche, attraverso un sistema di

templating del tutto compatibile con AWS CloudFormation ed AWS CloudWatch, pertanto nella direzione della portabilità verso altri cloud. Viene discussa anche l'orchestrazione tramite Juju, che rappresenta la soluzione ideale per l'erogazione di singoli servizi enterprise. Successivamente si è posta l’attenzione sulla PaaS di Openshift, un progetto open source promosso da Red Hat, il cui scopo è quello di fornire una piattaforma di sviluppo scalabile, distribuita ed estendibile, la quale esula il contesto classico delle PaaS vincolate ai provider, e permette di studiare delle soluzioni ad-hoc inter-cloud. Vengono discussi i componenti fondamentali dell'architettura, quali Broker, Nodi, Cartridge e Gear, in funzione della caratteristica di auto scaling delle applicazioni di cui viene effettuato il deploy nella piattaforma. Openshift fornisce dei runtime comuni per il deploy di applicazioni Web Enterprise quali JavaEE, Ruby, Python o PHP, ed inoltre, nella soluzione di cloud privato, è possibile estendere qualsiasi runtime grazie al concetto di Cartridge.

Infine, nel quarto capitolo, viene descritta una configurazione d'infrastruttura dimostrativa realizzata nel laboratorio di Grid computing (Gridlab), attraverso l'uso delle versioni RDO Openstack Grizzly su un nodo host, e delle istanze Openshift Origin orchestrate tramite Heat. Vengono analizzati i componenti necessari alla realizzazione dell'infrastruttura ed alla procedure volte a configurare un ambiente IaaS su base Openstack, con sistema di virtualizzazione totale assista dall'hardware basato su KVM, su base CentOS Linux. Viene illustrato un template che effettua l'orchestrazione di una coppia di nodi Broker/Nodo Openshift con configurazioni standard, ed attraverso un deploy dimostrativo, vengono illustrate tutte le componenti di Openstack ed Openshift coinvolte.

CAPITOLO 1. CLOUD COMPUTING

Cloud est omnis divisum in partes tres, quarum unam incolunt IaaS, aliam PaaS, tertiam SaaS appellantur [1]

Il paradigma del cloud computing è divenuto, nel corso degli ultimi anni, uno dei più interessanti trend nel campo del calcolo distribuito e nella condivisione delle risorse, riuscendo a trasformare concettualmente ed empiricamente sia la produzione di beni e servizi nel Information Technology, sia le attività di ricerca scientifica. Tale migrazione dalle consuete e consolidate metodologie di ricerca e produzione, è stato favorito dal modello di economia di scala alla base del cloud computing. Quest'ultimo ha avuto il merito principale, in un settore come quello del IT in costante divenire, di avvicinare i mezzi alle idee, attraverso un canale di comunicazione del quale non è necessario conoscerne fattezze e funzionamento, appunto "tra le nuvole", che permette l'accesso a delle risorse teoricamente illimitate in maniera semplice ed intuitiva. Il cloud pertanto supporta efficientemente la produzione industriale e la ricerca accademica, poiché permette l'esecuzione di attività e processi in breve tempo con l'utilizzo di risorse allocate *on-demand* [2].

Il termine cloud è preso in prestito dalle telecomunicazioni, quando agli inizi degli anni 90 le compagnie telefoniche offrirono servizi di VPN a banda garantita e basso costo, grazie alla possibilità di effettuare lo switch del traffico su diversi percorsi. Tale instradamento divenne appunto non deterministico, ovvero impossibile da calcolare determinare a priori, e da qui nacque il concetto di servizi in cloud, che coinvolge ormai tutto l'IT. Il concetto proprio di cloud computing viene espresso formalmente per la prima volta in (Chelappa, 1997) e si definisce come un paradigma di calcolo computazionale dove i limiti alla potenza di calcolo sono determinati da fattori economici in senso stretto, piuttosto che limiti tecnici. Tuttavia la definizione formale più completa ed accettata è quella di Foster, il quale afferma che il cloud è un paradigma

di calcolo distribuito di largo dominio, guidato da economia di scala, nel quale un insieme di potenze di calcolo controllate, archiviazione dati, piattaforme e servizi astratti, virtualizzati e dinamicamente scalabili sono rese disponibili on-demand a utenti esterni attraverso Internet. [3]

1.1 Tipologie di Cloud Computing

Il Cloud viene suddiviso logicamente in cinque componenti essenziali, tre modelli di deployment e tre modelli di servizio. [4]

1.1.2 Caratteristiche essenziali

Le cinque componenti essenziali sono illustrate in Fig. 1, e sono rispettivamente:
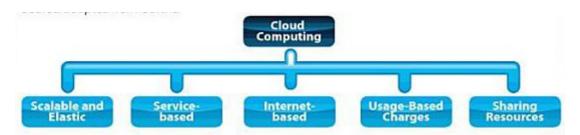
Fig. 1: Caratteristiche del Cloud Computing [4]

- **Scalabile ed Elastico**: nel paradigma del cloud computing, un provider non può prevedere l'uso dei volumi o la mole di richieste di accesso ai servizi; né tanto meno si deve far carico di problemi di tale natura. La peculiarità del servizio erogato attraverso il cloud è la sua costante disponibilità online, detto appunto *always-on* , progettato per essere altamente scalabile e flessibile. L'utente finale ha l'impressione di avere accesso ad una quantità di risorse potenzialmente illimitate, di cui può usufruire in qualunque quantità e tempo;

- **Service-based**: gli utenti possono disporre nel cloud di applicazioni, tempo di CPU, archiviazione dati o processi complessi come un servizio. Ed inoltre tale interazione avviene on-demand automaticamente senza l'intervento umano;
- **Internet-based**: Internet viene utilizzato come un mezzo di erogazione di servizi offerti sul Cloud, attraverso dei Web Service con interfacce standardizzate;
- **Costi a consumo** : nel paradigma del cloud è previsto il pagamento per la quantità di servizio erogato. L'utente percepisce l'accesso di una potenza illimitata, ma poi paga ciò che ha effettivamente utilizzato;
- **Risorse condivise:** I servizi cloud sono *multi-tentant*, ovvero esistono in un ambiente di lavoro condiviso da più entità, sia a livello hardware, che a livello software. Ciò può significare che una singola istanza di un software, e la piattaforma hardware dove viene eseguita, serve più client. Vi è un'indipendenza di locazione del software e quindi del servizio erogato, celata all'utente finale il quale può al più decidere una macro-area geografica di pertinenza ad un più alto livello di astrazione.

1.1.2 Modelli di Deployment nel Cloud

Il cloud viene percepito principalmente nella sua accezione più nota di servizio erogato al pubblico da pochi provider elitari, che dispongono di un'immensa quantità di risorse da orchestrare, attraverso il ben noto paradigma in esame. Tuttavia esistono tre modelli di deployment nel cloud:

- **Public Cloud:** il cloud pubblico è reso disponibile su Internet per utenti generici o grandi gruppi industriali, ed è tipicamente gestito da organizzazioni che erogano servizi cloud. Tali servizi vengono offerti attraverso la rete, ed esposti in maniera standardizzata, auto-configurante ed predisposti per il pagamento a consumo. Il cloud pubblico appartiene ad un mercato massivo, con una serie di servizi altamente standardizzati e con bassi margini; [5]

- **Private Cloud**: il cloud privato è operato esclusivamente per qualche organizzazione/entità . Viene gestito direttamente da tali organizzazioni o da terze parti con accordi di erogazioni di sevizi cloud esclusivi con queste. Al di là del substrato tecnologico, un'importante differenza fra i grandi data center delle corporation e i cloud privati risiede nel modello di gestione. Poiché nel modello cloud il business vero e proprio è il cliente, il cloud privato risulta essere essenzialmente un mercato di strumenti utilizzabili sotto licenza, di hosting e di consulenza;
- **Hybrid Cloud**: il cloud ibrido è un'infrastruttura ottenuta dalla combinazione dei cloud pubblici e privati, i quali rimangono entità distinte, ma collegate e amalgamate da tecnologie che garantiscono la portabilità di applicazioni e dati. Associa elementi di cloud pubblico e privato, inclusa qualsiasi combinazione di provider e consumer, e può contenere più livelli di servizio.
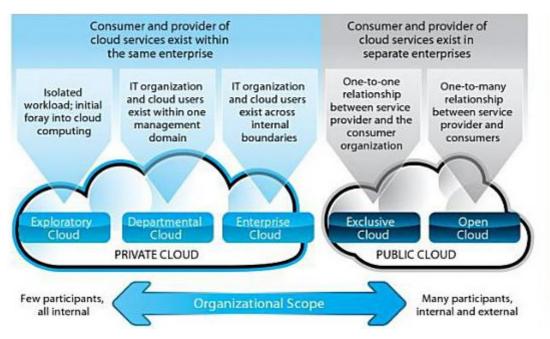
Fig.2 Modelli cloud offerti [4]

Il cloud computing, come illustrato in Fig. 2, definisce a sua volta altri sub-modelli all'interno dei modelli di cloud privato e pubblico. Queste segmentazioni, danno luogo ad una descrizione più concisa sui vari scenari che coinvolgono provider e consumer in un contesto cloud. Inoltre i sub-modelli implicano un livello di conoscenza e

specializzazione necessaria per garantire l'erogazione dei servizi cloud offerti.

Il Private Cloud contiene dunque a sua volta:

- **Exploratory cloud**: coinvolge le organizzazioni che vogliono puntare sul cloud, ma che hanno bisogno di acquisite conoscenza ed esperienza su questo settore dell' IT, a loro inizialmente sconosciuto. Il suo valore scaturisce dalla capacità di sviluppo di servizi cloud e dalla comprensione degli aspetti tecnici coinvolti in un ambiente cloud, come punto di partenza per determinare dei ricavi su investimenti fatti attraverso l'erogazione di servizi nel cloud;
- **Departmental cloud**: in questo modello, sia le organizzazioni che erogano servizi nel cloud, sia i dipartimenti che usufruiscono servizi nel cloud, appartengono alla stessa organizzazione. Nel cloud dipartimentale, lo scopo è quello di espandere l'uso del cloud computing verso unità di business ed operazionali. Il suo valore deriva da competenze avanzate su risorse condivise e gestione della virtualizzazione, insieme ad un generale miglioramento della conoscenza dei modelli offerti nel cloud;
- **Enterprise cloud**: come nel precedente modello, le organizzazioni di IT che erogano servizi nel cloud e quelle consumano tali servizi fanno parte della stessa unità enterprise, ma esistono diversi confini di gestione. Lo scopo nel cloud enterprise è quello di eseguire dinamicamente processi aziendali *time-critical* con risorse preesistenti precedentemente inutilizzate o non pienamente sfruttate. Il valore che fornisce questo modello è dato dall'ottimizzazione degli investimenti aziendali in termini di risorse e capacità tecnologiche. Uno degli aspetti cruciali in un contesto cloud è quello legato alla sicurezza ed il cloud enterprice è in questo senso ancora più restrittivo, poiché richiede che essa sia ben definita ed implementata, oltre alla possibilità per le organizzazioni di avere a disposizione la gestione di servizi con funzionalità addizionali, l'automazione del provisioning e la gestione dell'autenticazione. Inoltre fondamentale è la funzione di misura delle risorse utilizzate per garantire le operazioni di pagamento a consumo, peculiarità intrinseca nel cloud computing.

Per ognuno dei tre sub-modelli di cloud privato, sia l'unità di business che quella operazionale risiedono nello stesso dominio enterprice, erogati come servizi nel cloud. Per quanto riguardo invece il cloud pubblico, esso di suddivide ulteriormente in:

- **Exclusive Cloud**: tale modello è usato generalmente per fornire dei servizi condivisi a membri di un singolo gruppo o di una singola organizzazione, come ad esempio in consorzio universitario o più aziende con i loro relativi partner aziendali *trusted*. Lo scopo è quello di fornire un accesso a partecipanti fidati, ovvero trusted, al fine di garantire loro la possibilità di utilizzare le proprio applicazioni critiche. Gli utenti riconosciuti possono usufruire dei servizi attraverso la loro relazione con questa struttura organizzativa ad ombrello, fintanto che tale organizzazione ha degli accordi con il provider cloud. Le caratteristiche distintive critiche del cloud pubblico esclusivo sono due. La prima, è che il provider e l'organizzazione che usufruisce dei suoi servizi nel cloud sono note reciprocamente e sono in grado di negoziare dei parametri di livello di servizio. La seconda, invece, prevede che ogni risorsa fornita usata per erogare un servizio verso l'organizzazione, sia dedicata per questo cloud esclusivo e non condivisa con altri ambienti di delivery o altri consumer al di fuori dell'organizzazione stessa. E' importante notare a tal proposito, che un cloud esclusivo ed un cloud privato virtuale non sono la stessa cosa. Poiché quest'ultimo descrive un accordo dove una singola organizzazione consumer ha accesso esclusivo alle risorse del provider che fornisce i servizi del consumer. Tuttavia queste risorse, risiedono in un data center che fornisce servizi a molti altre organizzazioni consumer.

- **Open Cloud**: in questo modello, le organizzazioni che erogano e che usufruiscono dei servizi non sono note fra di loro prima della presentazione di una richiesta di servizio. La caratteristica principale di un cloud open risiede nel contrattazione dei servizi, la quale deve essere automatizzata, e nella definizione basata su degli standard degli eventi e nel controllo di essi da parte del provider. Il modello open cloud richiede inoltre l'automatizzazione della contrattazione e della riconciliazione dei livelli di servizi, ed inoltre delle

politiche di pricing e della policy. Generalmente i server sono disposti e pagati tramite Internet, spesso senza l'intervento umano che si riduce al minimo indispensabile. Il valore che si ottiene da questo tipo di cloud è l'eliminazione il capitale di investimento che il consumer dovrebbe disporre in anticipo per ottenere dei servizi e delle risorse interne alla propria unità privata simili a quelle a disposizione sul cloud, azzerando inoltre i tempi di implementazione e di realizzazione di tale unità, già pronta on-demand sul cloud.

1.1.3 Modelli di Servizio nel Cloud

Oltre alla generica definizione di Cloud Computing, la quale dà l'idea generale del paradigma di calcolo computazionale on-demand, il cloud contiene in sé ulteriori tre paradigmi, indicati formalmente come *Cloud Service types*, che suddividono logicamente i contesti di riferimento dei vari cloud, organizzati in una struttura piramidale a tre livelli. Tale piramide è detta di Sheehan [6], dal nome del suo ideatore, ed è definita come segue :

Fig. 3: La piramide del Cloud

- **SaaS (Software as a Service)**: in questo livello il software è presentato agli utenti finali come un servizio on-demand, tipicamente come un servizio disponibile sul web. L'utente che usufruisce di tale servizio, non si deve curare di problemi quali deploying e mantenimento del software, ma ottiene in maniera semplicificata ed intuitiva il servizio stesso, il quale viene gestito in maniera condivisa nel cloud, con un meccanismo di scaling che permette l'aggiunta immediata di funzionalità addizionali all'occorrenza (on-demand), in modo del tutto trasparente. Esempi pratici sono le Google Apps, Salesforce.com, Wordpress.com, sebbene ormai la quasi totalità dei servizi offerti in rete ricada in questa definizione;

- **PaaS (Platform as a Service):** fornisce una piattaforma di sviluppo software con una serie di servizi e funzionalità che supportano e assistono le applicazioni realizzate nel cloud durante tutto il ciclo di vita dello sviluppo. Le fasi di progettazione, sviluppo, testing, deploy, monitoraggio e hosting sono tutte coivolte e pensate nel cloud, ottenendo così un'ambiente di sviluppo accessibile ed utilizzabile online, anche in maniera collaborativa per team distribuiti geograficamente in parti diverse. Questo livello è rivolto agli sviluppatori ed è anche noto pertanto come Cloudware. Gli sviluppatori realizzano le loro applicazioni secondo alcune specifiche e successivamente caricano il loro codice nel cloud. La particolarità interessante di questo approccio è che l'applicazione, che viene eseguita nel cloud, è generalmente in grado di scalare automaticamente in base alla mole di utenti che interagiscono con essa. Esempi pratici sono Heroku, App Engine, Openshift, Joyent, Force.com.

- **IaaS (Infrastructure as a Service):** questo livello si fa carico della virtualizzazione della potenza di calcolo, dello storage e della connettività di rete di più data center, offrendo all'utente la possibilità di gestire un'infrastruttura di elaborazione completa in rete. È noto anche come HaaS (Hardware as a Service), ed è il tipo di cloud più potente e più flessibile poiché permette a sviluppatori ed amministratori di sistema la massima libertà di scelta dei componenti hardware (virtualizzati) e di scaling delle risorse , il tutto fornito sempre come un servizio offerto nel cloud, pertanto on demand. Inoltre, generalmente, più *tenant* coesistono sulle stesse risorse di infrastruttura. Esempi pratici sono Amazon EC2, pioniere e leader indiscusso nel settore, Openstack, VMware vCloud Hybrid Service, Microsoft Azure.

La piramide del Cloud esprime in maniera esplicativa e chiara la suddivisione dei vari cloud, ed offre inoltre una visione della mole di utenti potenzialmente interessata da un certo livello. Come illustrato in Fig.2 per mezzo di una piramide rovesciata che la rappresenta la distribuzione dei provider nel cloud, i cloud SaaS sono quelli più ricorrenti e più utilizzati, mentre man mano che si scende di livello, diminuisce il target di utenti coilvolti, pertanto il numero di provider.
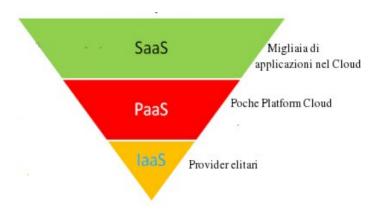
Fig. 4 Numero di provider nel Cloud

È possibile, dunque, dedurre un'ulteriore suddivisione dei tipi di cloud, in termini di grado di libertà d'azione:

- **Cloud Application**: fa riferimento al livello SaaS e restringe il campo d'azione degli utenti finali i quali possono solo usufruire del servizio, senza possibilità di personalizzazioni o aggiunta di feature non previste. Non vi è alcun controllo o conoscenza delle tecnologie che stanno al di sotto dell'applicazione.
- **Cloud Platform**: è corrispondente al livello PaaS, e sebbene dia la possibilità agli sviluppatori di caricare le proprio applicazioni, tuttavia essere rimangono vincolate alla piattaforma stessa, ottenendo un campo d'azione limito all'envoriment della platform.
- **Cloud Infrastructure**: combacia con il livello IaaS, dà la massima libertà nel contesto cloud, garantendo il controllo totale dell'infrastruttura server, non confinata in termini di container, application o istanze restrittive.

1.2 Architettura Cloud

Il Cloud possiede una propria architettura, che è strutturata in funzione del suo modello di business. Contrariamente al modello tradizionale, esso prevede il pagamento del fornitore del servizio (provider) in base al consumo e quindi in base alla risorse utilizzate per l'erogazione del servizio stesso, in maniera continuativa e auto-scalabile. Tipicamente, viene definito un prezzo in termini di consumo di CPU , di quantità di banda e di storage nell'unità di tempo, dando all'utente la possibilità di ottenere maggiori risorse per carichi di lavoro o burst maggiori, senza avere l'obbligo di mantenere attive tali risorse qualora non strettamente necessarie.
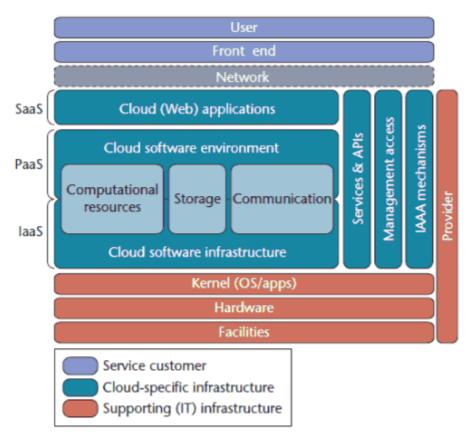
Fig. 5: Architettura Cloud [11]

In base a queste premesse, si evince che il Cloud ha delle similarità con il concetto di Griglia computazionale. Le griglie, utilizzate prevalentemente in ambito scientifico e di ricerca, prevedono grandi potenze di calcolo ed una rete di risorse condivise dove i risultati dei processi immessi nella griglia vengono a loro volta condivisi fra i processi che partecipano al grid computing. I partecipanti al grid computing, accedono attraverso una serie di protocolli dedicati, e generalmente poco standardizzati. I cloud diversamente, fanno riferimento ad un'imponente mole di risorse di calcolo, di rete e di storage, alle quali è possibile effettuare l'accesso tramite protocolli standard come Web Service RESTful. Il vantaggio ulteriore consiste nel fatto che è possibile implementare i Cloud su un substrato tecnologico preesistente di griglie computazionali. Difatti, osservando l'immagine in Fig. 6, il Cloud Computing si sovrappone alle griglie computazionali, evolvendo in un certo senso al di fuori di esse ed utilizzandole come dorsale e infrastruttura di supporto per le risorse, le quali possono essere così disponibili

per mezzo di protocolli standard attraverso un'interfaccia astratta.
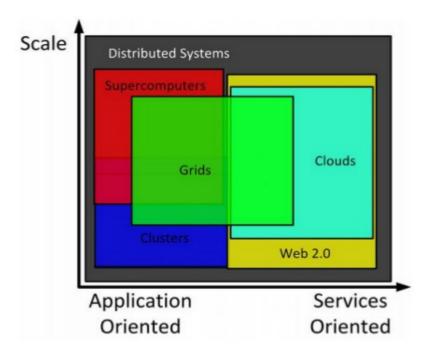
Fig. 6: Overview tipi di sistemi distribuiti

Entrambi i paradigmi ambiscono ad offrire risorse di calcolo astratte, tuttavia differiscono nel loro particolare approccio ed oggetto:

- Il Cloud computing punta a servire più utenti nello stesso tempo, mentre le griglie computazionali tendono a fornire funzionalità nell'ordine di grandezze e di qualità dei Supercomputer, attraverso un meccanismo di coda dei processi;
- Nelle griglie le risorse appartengono e sono operate da più provider, mentre nel cloud sono tipicamente sotto il controllo di un'unica organizzazione;
- I servizi Cloud possono essere ottenuti utilizzando interfacce standardizzate su una rete, mentre le griglie richiedono l'esecuzione del software Grid fabric localmente.

Pertanto, l'architettura Cloud è simile a quella Grid, come si evince in Fig. 7, e vi è un collegamento logico fra i livelli infrastrutturali con i servizi Cloud [7], esposti precedentemente nella piramide di Sheehan.
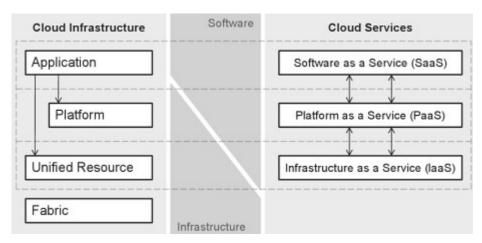
Fig. 7 Architettura Cloud in relazione ai servizi Cloud [7]

- **Fabric**: Sebbene appartenga all'architettura, il livello Fabric è quello escluso dai servizi Cloud, poiché contiene le risorse hardware fisiche, quali CPU, memorie e risorse di rete;
- **Unified Resource**: questo livello contiene le risorse che sono state astratte e/o incapsulate, tipicamente tramite virtualizzazione, e che possono essere esposte ai livelli superiori;
- **Platform**: aggiunge una serie di tool, servizi e middleware sul livello inferiore per fornire un ambiente di sviluppo in una Platform;
- **Application**: contiene le applicazioni che vengono eseguite nel Cloud.

1.2.1 Modello computazionale

Cloud e Grid differiscono nel loro rispettivo modello computazionale. Le griglie

solitamente usano un modello computazionale batch-scheduled, nel quale un Local Resource Manager come Condor, gestisce delle risorse di calcolo per un sito nella griglia. Gli utenti poi richiedono l'esecuzione dei propri task di lavoro (job) attraverso GRAM per richiedere l'accesso a qualche risorsa per un certo intervallo di tempo. Contrariamente, il cloud computing prevede che le risorse siano condivise fra tutti gli utenti allo stesso tempo, escludendo un sistema di gestione della coda delle richieste di esecuzioni di processi come avviene nelle griglie. Pertanto le applicazioni più sensibili ai tempi di risposta traggono maggiore beneficio nell'esecuzione nel cloud, poiché l'infrastruttura è progettata ed offerta al fine di garantire servizi di questo tipo, in base ad un organizzazione logica che prevede la scalabilità delle applicazioni in base al numero di utenti. Difatti, a causa dei tempi di scheduling, si staging e di coda dei job, molti ambienti Grid non supportano nativamente l'interazione fra le applicazioni. Ad esempio, se un job richiedesse 60 minuti su 100 processori, lo stesso rimarrebbe in attesa nella coda del LRM fino a quando i 100 processori richiesti non siano disponibili per i 60 minuti richiesti, allorché tale potenza di calcolo sarebbe stata allocata e dedicata per tutta la durata del job.

1.2.2 Modello Dati

Trascurando delle sommarie e frettolose previsioni secondo le quali il calcolo computazionale erogato attraverso Internert (Internet Computing) verrà inglobato e amministrato nella sfera del Cloud Computing attraverso un modello centralizzato, la previsione più verosimile è quella fornita da Foster, secondo la quale l'Internet Computing sarà centralizzato e focalizzato sui dati, e il Cloud Computing, insieme al Client Computing, coesisteranno ed evolveranno di pari passo alla crescita costante di applicazioni che fanno uso intensivo di dati. Questo perché sebbene si vada sempre più nella direzione del cloud computing, il client computing riveste sempre una vitale importanza poiché gli utenti finali potrebbero voler utilizzare le funzionalità hardware

di cui hanno bisogno i loro processi locali.
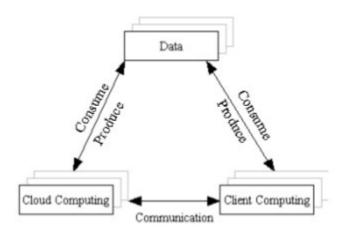
Fig. 8 Il modello triangolare dell'Internet Computing [3]

Nel contesto delle griglie computazionali, i dati rivestono un'importanza strategica, l'esistenza di numerosi progetti sul Data Grid ne è una palese testimonianza. In particolare, è stato introdotto il concetto di dato virtuale, il quale ha svolto un ruolo cruciale. Un dato virtuale è sostanzialmente un'entità che unisce logicamente i dati, i programmi e le computazioni e che fornisce una serie di astrazioni quali trasparenza della locazione dei dati, un catalogo di metadati distribuiti. Sia nel cloud, sia nelle griglie, diviene critica la gestione della combinazione delle risorse di calcolo con quelle relative ai dati. Poiché appunto l'accesso ai dati può diventare un collo di bottiglia, tutti gli sforzi sono concentrati verso lo studio di pattern di accesso ai dati che minimizzano il numero di spostamenti di dati e, pertanto, aumentino le performance e la scalabilità delle applicazioni. Da qui la necessità di un'unificazione dei problemi computazionali e di archiviazione dei dati, volta a ridurre lo spostamento dei dati e quindi strategico è il concetto di località dei dati. E sebbene le griglie computazionali abbiamo presentato contributi significativi in questa direzione nel corso tempo, lo scenario di riferimento dove i pattern di accesso ai dati e gli scheduler *data-aware,* è proprio il cloud, o meglio i cloud, dove tali politiche sono combinate nel migliore dei modi per garantire il corretto funzionamento delle applicazioni, pronte a scalare all'occorrenza.

1.2.3 Virtualizzazione

La virtualizzazione è uno degli aspetti essenziali nel cloud computing; difatti è principalmente grazie ad essa, che i consumer, o utenti finali, hanno l'illusione di un ambiente dedicato esclusivamente per le loro applicazioni, ed è sempre essa che svolge un ruolo critico nel concetto di scalabilità. In termini tecnici, la virtualizzazione fornisce la necessaria astrazione affinché le unità fisiche appartenenti al livello Fabric sottostante, possano essere aggregate come un insieme di risorse, potendo infine, a livello ancora più astratto, esporre strumenti di utilità per l'accesso alle risorse stesse.

Un'altra caratteristica cruciale, che viene inoltre associata al concetto stesso di cloud, è la possibilità grazie alla virtualizzazione di incapsulare le applicazioni, potendo gestire completamente il loro ciclo di vita, con un alto livello di astrazione. Le operazioni di configurazione, deploy, avvio, migrazione, sospensione, ripristino o terminazione delle applicazioni sono gestite tutte in maniera trasparente, garantendo inoltre un alto livello di sicurezza, manutenzione ed isolamento del software.

Oltre ad offrire la computazione on-demand, la virtualizzazione permette di ottenere dei sistemi più efficienti in termini di consolidamento, la quale viene effettuata in maniera del tutto trasparente agli utenti dei servizi cloud. La *server consolidation*, permette di aggregare logicamente sulla stessa risorsa fisica più risorse virtuali, garantendo l'isolamento ed aumentando la percentuali di utilizzo di risorse che potrebbero essere altrimenti non sfruttate a pieno. E se la capacità delle risorse fisiche è sufficiente, la condivisione sarà del tutto trasparente.

1.2.3.1 Hypervisor

La virtualizzazione, storicamente promossa ed adottata per primo da VMware, ha introdotto un considerevole aumento di efficienza all'interno dei data center. L'elemento chiave che sta alla base della virtualizzazione è denominato Hypervisor o Virtual Machine Monitor (VMM) di tipo 1, e corrisponde ad un livello intermedio in mezzo ai sistemi fisici e logici, il quale "inganna" il sistema operativo soprastante, facendogli credere di avere accesso diretto a delle risorse hardware in realtà virtuali. In questo modo, i sistemi operativi, contenuti all'interno di macchine virtuali (VM), hanno un accesso trasparente alle risorse.
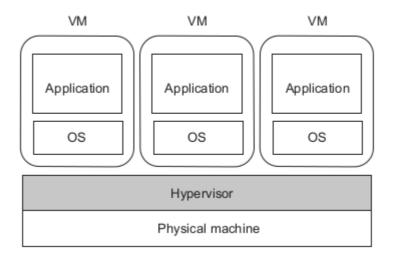
Fig. 9: Virtualizzazione tramite Hypervisor

Un Hypervisor permette di eseguire più sistemi operativi contemporaneamente sullo stesso insieme di risorse hardware, svolgendo una funzione anche di monitoring delle istanze virtuali eseguite nel suo contesto. Attualmente, gli hypervisor più diffusi sul mercato, sono ESX di VMware, XenServer di Cytrix, KVM, Hyper-V di Microsoft, ed il mercato è in continua espansione grazie alla crescita costante degli investimenti nel cloud computing. Inoltre, grazie all'introduzione di tecnologie all'interno dei processori, appositamente create per ottimizzare la virtualizzazione, quali Intel VT e AMD-V, i livelli di performance ottenute con macchine virtuali sono paragonabili a quelle con hardware fisico, il che rende il risultato di estrema importanza in un ambiente

fortemente incentrato sulla virtualizzazione come quello del cloud.

Secondo la definizione di Goldberg [8], vi sono due tipi di hypervisor il primo è denominato Bare-Metal Hypervisor (Tipo 1), e viene eseguito direttamente sull'hardware che controlla, e di cui gestisce i sistemi operativi guest soprastanti. Ne fanno parte gli hypervisor citati precedentemente; il secondo prende il nome di Hosted Hypervisor (Tipo 2), ed è in esecuzione all'interno di un sistema operativo. Esempi pratici sono VMware Server e Virtualbox. [9] La Fig. 10 illustra tale suddivisione.
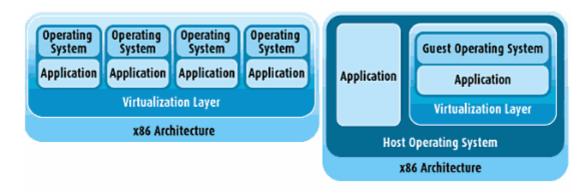
Fig. 10: Hypervisor Bare-Metal e Hosted [9]

La famiglia di processori x86 prevede un insieme di livelli di protezione, organizzati concettualmente ad anelli (da cui prendono il nome in gergo tecnico di *Ring*), nei quali poter eseguire il codice macchina. I livelli sono ordinati in base ai privilegi di esecuzione e vanno da 0 a 3. Il codice delle applicazioni utente è eseguito generalmente nel Ring 3, mentre quello relative ad istruzioni privilegiate del sistema operativo, le quali hanno bisogno di avere accesso diretto a memoria e hardware, risiede nel Ring 0 (ad esempio codice in system space, kernel mode e supervisor mode). Nella virtualizzazione, l'hypervisor è un programma e risiede nel Ring 0. Il suo compito è quello di gestire risorse e allocazione della memoria per la macchine virtuali, ma anche di fornire delle interfacce accessibili da livelli più alti, per il monitoraggio e la gestione dei sistemi virtuali.

Poiché come accennato, i sistemi operativi eseguono le loro operazioni privilegiate al Ring 0, si pone il problema del declassamento di livello di tali operazioni rispetto a quelle del hypervisor, e ciò non è direttamente possibile. La soluzione per sopperire a

questo problema avviene mediante tre tecniche: virtualizzazione totale (full), para-virtualizzazione e virtualizzazione hardware.

1.2.3.2 Virtualizzazione totale con traduzione binaria

La virtualizzazione totale (Full virtualization) utilizza una combinazione di esecuzione diretta e traduzione binaria, attraverso la quale avviene una traduzione del codice a livello kernel per rimpiazzare delle istruzioni non virtualizzabili (quelle scritte per essere eseguite nel Ring 0) con nuove sequenze di istruzioni, che hanno effetto sull'hardware virtualizzato. Allo stesso tempo, il codice del livello utente (Ring 3) è eseguito direttamente sul processore, ottenendo delle ottime prestazioni di virtualizzazione.
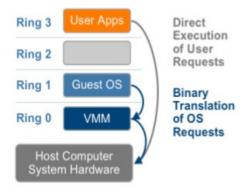
Fig. 11: Traduzione binaria nella virtualizzazione totale [10]

Con questo approccio, il sistema operativo guest viene astratto in modo completo dall'hardware sottostante attraverso l'hypervisor risiedente al Ring 0. Non richiede il supporto di hardware o di sistemi operativi predisposti alla virtualizzazione, poiché l'hypervisor traduce tutte le istruzioni del sistema operativo a runtime; mantiene una cache dei risultati per velocizzare istruzioni ripetitive. Con questa soluzione però, si

ottengono delle degradazioni delle prestazioni in termini di velocità, rispetto alle altre elencate di seguito, per via dell'overhead generato dalla traduzione delle istruzioni.

1.2.3.3 Paravirtualizzazione

La paravirtualizzazione è una tecnica che prevede la virtualizzazione assistita dal sistema operativo, ovvero che coinvolge il sistema operativo guest nella comunicazione con l'hypervisor, per effettuare delle chiamate di sistema speciali dette "hypercall", le quali, come illustrato in Fig.11, permettono di eseguire operazioni critiche altrimenti possibili solo Ring 0 (non virtualizzabili). Questo approccio permette di raggiungere delle performance migliori in termini di velocità, grazie all'eliminazione del processo di traduzione binaria, ma costringe a delle modifiche nel kernel dei sistemi operativi per la virtualizzazione delle risorse da utilizzare.
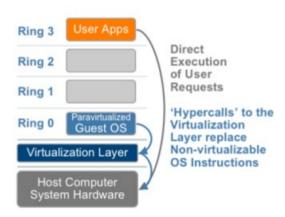
Fig. 12: Paravirtualizzazione [10]

1.2.3.4 Virtualizzazione hardware-assisted

La virtualizzazione assistita dall'hardware usufruisce delle estensioni alla

virtualizzazione introdotte nei moderni processori quali Intel VT e AMD-V, come osservato precedentemente. Tali estensioni, permettono di eseguire delle macchine virtuali come se fossero in virtualizzazione totale, senza la necessità della traduzione binaria grazie all'aggiunta di un'ulteriore livello di privilegi, al di sopra del Ring 0, denominato Root Mode. In questa modalità, l'hypervisor può operare lasciando il Ring 0 a disposizione dei sistemi operativi, che non necessitano di alcuna modifica.

Questo tipo di virtualizzazione supera le altre due soluzioni analizzate ed è diventa la scelta de facto per le soluzioni di virtualizzazione bare-metal. Tuttavia non è possibile esprimere un giudizio universale sulle performance di questo approccio rispetto alla paravirtualizzazione perché è sensibile al contesto.
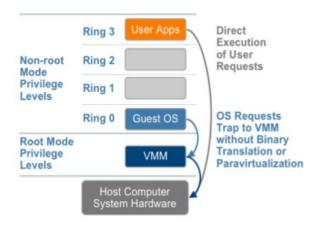
Fig. 13:

Virtualizzazione hardware-assisted [10]

La paravirtualizzazione spesso risulta più veloce, ma non non garantisce un contesto totalmente isolato, pertanto è necessario analizzare dei criteri di valutazione per determinare una scelta su quale possa essere la migliore. Di seguito sono discussi alcuni degli hypervisor più comuni utilizzati nel cloud computing, che appartengono alle categorie di virtualizzazione appena elencate.

1.2.3.5 Xen

Xen è un'iniziativa open source che implementa la paravirtualizzazione tramite Xen Hypervisor. Come analizzato nel paragrafo 1.2.3.3, lo Xen Hypervisor viene eseguito

nel livello con privilegi di esecuzione più alti, e controlla gli accessi del sistema operativo guest all'hardware sottostante. Per mezzo di un ulteriore divisione concettuale, i sistemi operativi guest vengono eseguiti in Domini, i quali rappresentano le istanze delle macchine virtuali. Seguendo la stessa logica, viene eseguito nel Dominio 0 uno speciale software di controllo utilizzato per accedere alla macchina host (dove risiede l'hypervisor) e per controllare i sistemi operativi guest. Questo componente viene avviato dopo il caricamento dell'hypervisor ed espone un interfaccia HTTP per la gestione delle macchine virtuali, dai livelli superiori.

La virtualizzazione è implementata eseguendo l'hypervisor al Ring 0, nonché Dominio 0, e tutti i sistemi operativi guest nel Ring 1, denominato Dominio U; mentre le applicazioni continuano a risiedere nel Ring 3. Tale scenario permette di mantenere invariata la Application Binary Interface (ABI), risultando trasparente alle applicazioni.

A causa della necessità strutturale, per questo approccio di virtualizzazione, di dover modificare il kernel del sistema operativo guest, Xen è generalmente disponibile e offerto su sistemi Linux, poiché open source e quindi liberamente modificabili, mentre non si presta al supporto su sistemi operativi proprietari.

1.2.3.6 VMware ESXi

ESXi è la nuova architettura di hypervisor bare-metal di VMware, indipendente dal sistema operativo. [11] L'architettura comprende il sistema operativo POSIX VMkernel, che viene eseguito sugli host ESX, e che funge da tramite fra le macchine virtuali e le risorse fisiche. La Fig. 14 illustra lo schema logico completo:
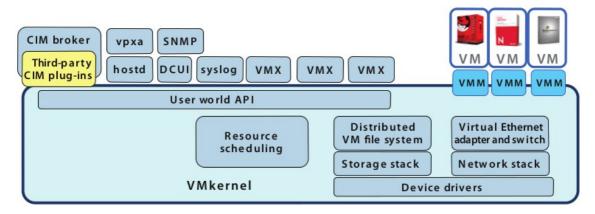
Fig. 14: Architettura di ESXi [11]

Il sistema Vmkernel è stato progettato per supportare più macchine virtuali in esecuzione, fornendo loro tutte le funzionalità basiche dei sistemi operativi quali creazione dei processi, allocazione della memoria, scheduling delle risorse, I/O, e gestione dei driver dei dispositivi. Inoltre è diskless, non richiede un particolare file system per la sua esecuzione, bensì contiene un file system in memoria utilizzato soltanto per le configurazioni ed i log; difatti viene fornito principalmente come firmware. I file system VMFS possono invece essere usati su storage esterni per le macchine virtuali, come avviene nella prassi dei sistemi ESX.

La novità dell'architettura, rispetto alle precedenti, è il sistema di User world API . Per User world vengono intesi i processi eseguiti in VMkernel, e poiché questo sistema contiene un sottoinsieme delle funzionalità tipiche dei sistemi POSIX, viene fornito un framework per i processi che necessitano di essere eseguiti in un livello di hypervisor, accessibile tramite le suddette API.

1.2.3.7 Microsoft Hyper-V

Hyper-V è un'infrastruttura di virtualizzazione realizzata da Microsoft, basato su hypervisor bare-metal con virtualizzazione assistita dall'hardware. Nato sulla versione a 64 bit di Windows Server 2008, è attualmente disponibile per la stessa architettura su Windows Server 2012, installabile come Role (discussi nel secondo capitolo). Hyper-V supporta l'isolamento in termini di partizioni, organizzate ad albero. Una partizione è un'unità logica di di isolamento delle macchine virtuali, supportata dall'hypervisor, nella quale viene eseguita il sistema operativo. La Fig. 15 mostra l'architettura di Hyper-V.
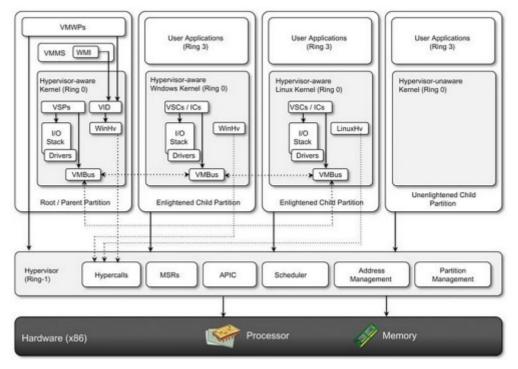
Fig. 15: Architettura Hyper-V

È prevista l'esistenza di almeno una partizione, denominata Root, la quale è l'unica ad avere accesso all'hardware. La partizione Root esegue lo stack di virtualizzazione, mantiene tutti i driver per configurare i sistemi operativi guest, e crea, grazie alle API delle hypercall, delle partizioni figlie per mezzo dell'hypervisor. Le stesse, sono utilizzate per contenere i sistemi operativi e non hanno accesso diretto all'hardware sottostante. L'interazione delle partizioni con l'hardware è effettuata per mezzo della partizione Root o dell'hypervisor direttamente. [12]

Come accennato, le partizioni non hanno accesso al processore fisico, bensì hanno una vista virtuale del processore ed eseguono i processi dei loro sistemi operativi in una regione di indirizzi di memoria, che è privata per ogni la partizione. Le partizioni figlie, che non hanno accesso diretto alle risorse hardware, hanno una vista virtuale delle risorse, denominate VDevs. Le richieste di interazione con le risorse virtuali vengono smistate alla partizione Root, tramite un bus logico denominato VMBus, che gestisce le richieste ed utilizza lo stesso bus per i risultati, in modo del tutto trasparente al sistema operativo.

1.2.3.8 KVM

Kernel-based Virtual Machine (KVM) è una soluzione open source di virtualizzazione hardware-assisted per il kernel Linux, ma estendibile ad altri sistemi operativi open source, come ad esempio SmartOS citato nel capitolo 2. Vi sono alcune discussioni teoriche sul tipo di hypervisor a cui appartiene KVM, ma è comunemente riconosciuto come hypervisor bare-metal perché capace di fornire una virtualizzazione totale assistita dall'hardware.

Come già accennato, la virtualizzazione assistita dall'hardware aggiunge due modi di esecuzione al processore, il quale cambia la sua modalità di esecuzione in base al tipo di istruzione che identifica; in questo modo il programma utente può utilizzare l'hypervisor per eseguire il suo codice. Nel caso di Intel, le modalità di esecuzione sono denominate operazioni root e non root Virtual Machine eXtension (VMX). Le operazioni del sistema operativo guest sono le VMX non root, e qualora, in questa modalità, vi sia un tentativo di esecuzione di istruzioni privilegiate, il processore cambia la modalità di operazione in VMX root, in cui l'hypervisor esegue le istruzioni. Questa transizione viene chiamata VM Entry, mentre il passaggio inverso, prende il nome di VM Exit. Il compito di KVM è proprio quello di gestire le VM Exit e di eseguire le istruzioni in VM Entry.

KVM è un modulo del kernel Linux, presente dunque a livello kernel-space, che non può creare macchine virtuali senza un'interazione user-space. Solitamente questo compito è svolto da Quick EMUlator (QEMU), un hypervisor di tipo 2 che si integra perfettamente con altre soluzioni di virtualizzazione per offrire una virtualizzazione totale, eventualmente assista dall'hardware. Normalmente, QEMU effettua un emulazione software dell'hardware con traduzione binaria, ma l'integrazione con KVM permette di fare eseguire al processore le istruzioni in contesti VMX non root, mentre quelle che non possono essere eseguite direttamente vengono identificate ed emulate da QEMU. La Fig. 16 illustra il processo di emulazione con l'integrazione di KVM.
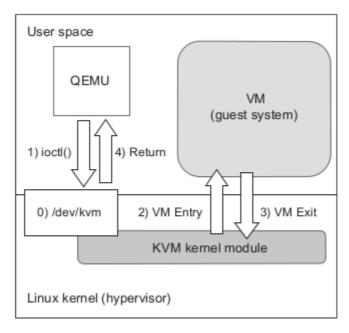
Fig. 16: Flusso di esecuzione QEMU/KVM [13]

Quando KVM viene caricato come modulo del kernel, viene creato il device virtuale /dev/kvm utilizzato da QEMU per le richiesta all'hypervisor. All'avvio del sistema operativo guest, QEMU effettua delle chiamate di systema ioctl()sul device virtuale per inizializzare il caricamento del sistema da parte del modulo KVM. Successivamente, il modulo effettua una VM Entry, iniziando ad eseguire il sistema operativo guest. Quando il sistema ha bisogno di eseguire delle istruzioni privilegiate, viene effettuata una VM Exit, e KVM identifica il tipo di istruzione che ha determinato il cambio di contesto. Infine il controllo viene trasferito nuovamente a QEMU che esegue le sue operazioni e termina l'operazione precedente con un'altra ioctl(), permettendo al processo di ricominciare. [13]

KVM viene solitamente fornito insieme ad altri componenti integrativi, che insieme permettono di realizzare il processo di virtualizzazione in modo trasparente. Uno di questi è libvirt, una libreria per la gestione di piattaforme di virtualizzazione tramite delle API. Libvirt supporta i più comuni hypervisor, quali KVM, Xen o ESX, il cui scopo è quello di fornire uno strato stabile e comune per gestire i domini di un nodo, possibilmente remoto. In libvirt è presente il concetto di dominio, come illustrato in Fig. 17, il quale rappresenta un'istanza di un sistema operativo in esecuzione su un

macchina virtuale fornita dall'hypervisor.
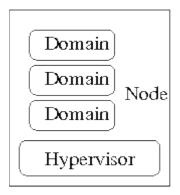
Fig. 17: Architettura logica in libvirt [14]

Un nodo è una singola macchina fisica, nel quale un hypervisor permette di virtualizzare più domini, quindi più macchine virtuali. Libvirt fornisce delle API che permettono di di gestire tutto il ciclo di vita delle istanze, sulle quali convergono i diversi hypervisor, a seconda della loro supporto a determinate funzionalità. Le API possono essere utilizzate su più domini, ma a partire da un singolo nodo fisico. L'interazione di KVM con libvirt è mostrata nella Fig. 18.
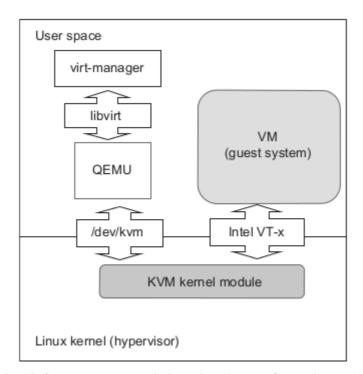
Fig. 18: Schema completo di virtualizzazione su QEMU/KVM [13]

Libvirt fornisce un driver multi-istanza per QEMU/KVM, a sua volta categorizzato un unico driver di sistema privilegiato, e più driver non privilegiati di sessione utente, permettendo l'accesso alle istanze per mezzo di un URI, del tipo: qemu:///session

1.2.3.9 Virtualizzazione nel cloud

Vi sono altre ragioni per le quali la virtualizzazione riveste un ruolo strategico nel cloud computing, le principali sono:

- ottimizzazione delle risorse utilizzate da più applicazioni sullo stesso server, in virtù del concetto di consolidazione di applicazioni e server;
- configurabilità e gestione automatizzata delle richieste di accesso a diverse risorse di calcolo o di storage, che grazie alla virtualizzazione diviene dinamica e scalabile all'occorrenza;
- aumento della disponibilità delle applicazioni, le quali possono essere sospese o ripristinate al bisogno, insieme a tutto il sistema in cui vengono eseguite, eseguendo delle semplici *snapshot* delle istanze in esecuzione;
- tempi di risposta minori, grazie all'automazione delle operazioni di gestione e di controllo, e al caching delle risorse utilizzate.

Da queste considerazioni, si evince come la virtualizzazione fornisca la possibilità di garantire stringenti SLA richiesti sovente in contesti aziendali, ottenendo anche una riduzione dei costi rispetti alle stesse classi di servizio in ambienti non virtualizzati, dove non è possibile raggiungere tali livelli di astrazione che ottimizzino l'intero processo industriale.

1.2.4 Sviluppo su Cloud Computing

Lo scopo del cloud computing è quello di offrire servizi ad un costo proporzionale alla quantità di computazione, archiviazione e banda erogata nell'unità di tempo. E la caratteristica principale dei cloud è quella di poter esporre servizi dinamicamente scalabili, insieme a risorse virtualizzate. Lo sviluppo di tali servizi, realizzato tenendo conto del punto di vista dell'utente finale, fa riferimento alle caratteristiche che un'infrastruttura di cloud computing possiede, schematizzata in Fig. 19.
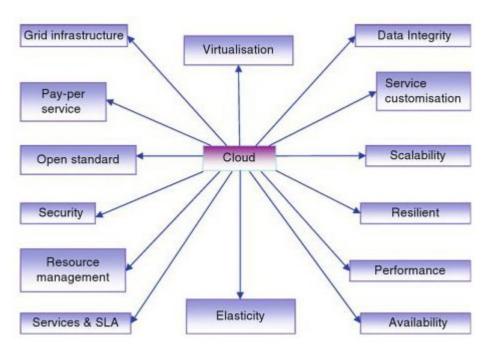
Fig. 19: Caratteristiche del cloud computing [15]

La conoscenza di queste caratteristiche è essenziale al fine di poter realizzare un servizio nel cloud e per monitorare il suo ciclo di vita. I Cloud Service sono solitamente esposti come Web Service, i quali seguono degli standard di descrizione, comunicazione e di discovery. Esempi pratici sono WSDL, SOAP, REST, UDDI. L'organizzazione e la gestione di questi servizi all'interno del cloud avviene all'interno di una Service Oriented Architecture (SOA). Un insieme di Cloud Service inoltre potrebbe essere utilizzati in un ambiente di sviluppo SOA, essendo così disponibili su varie piattaforme distribuiti e disponibili ulteriormente attraverso la Internet. I servizi che coinvolgono lo

storage sono distribuiti, accessibili, affidabili, e quelli dedicati al calcolo computazionale, che rappresentano il Service oriented computing, sono tipicamente soggetti a monitoring ed orchestrazione. Entrambe le tipologie di Cloud Service condividono delle caratteristiche fondamentali:

- Riusabilità dei Web Service e dei Core Service;
- Integrazione con servizi Enterprise;
- Supporto a Dynamic Binding e configurazione a runtime;
- Publishing, Subscribing e Discovery;
- Predisposizione ad interoperabilità con sistemi o componenti software di terze parti;
- Quality of Service;
- Affidabilità;
- Sicurezza.

Coloro i quali abbiano intenzione di sviluppare software ed architetture nel Cloud, devono tenere conto di queste caratteristiche per garantire che la progettazione dei componenti abbia un riscontro effettivo nell'ambiente di esecuzione, ottenendo dei servizi pienamente integrati, dinamici, scalabili e sicuri. Generalmente per Cloud Service si intende un servizio nel livello SaaS, a cui possono appartenere più Web Service, nell'accezione più classica del termine. [15]

Come discusso precedentemente, le applicazioni nel Cloud devono il loro successo al modello di deploy in cui esse sono coinvolte, potendo offrire servizi efficienti, erogati in base al consumo. Le metodologie classiche di progettazione del software non sono sufficienti ad interpretare il processo di sviluppo nel paradigma del cloud computing, in particolare nel modello di servizio SaaS. Pertanto, risulta necessario identificare le problematiche relative alle caratteristiche del cloud già nella fase di progettazione, a partire dalle specifiche del software che si vuole realizzare. La chiave della progettazione è l'individuazione delle funzioni di business a livello SaaS. Una volta

individuato il Business Process Management (BPM), si predispongono tali funzioni ad eventuali possibilità di personalizzazione da parte degli utenti finali (consumer). L'intero processo, schematizzato in Fig. 20, si riassume in essenzialmente in:

- Individuazione BPM;
- Individuazione SLA;
- Specifica di requisiti *service-oriented;*
- Specifica di requisiti di sicurezza.
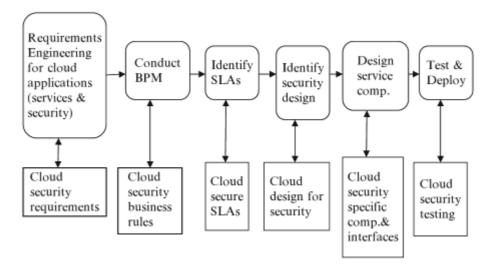
Fig. 20: Processo di sviluppo di applicazioni nel Cloud

1.2.5 Sicurezza

In virtù delle precedenti argomentazioni, risulta evidente come la sicurezza rivesta un

ruolo indispensabile nel paradigma del cloud computing. In ogni modello di servizio (SaaS, PaaS, IaaS), la sicurezza è parte integrale di applicazioni, reti, servizi, server, risorse e architetture cloud. Un aspetto che coinvolge non solo l'infrastruttura, il livello più basso nella piramide di Sheehan, ma anche il livello delle applicazioni dove sono in esecuzione più servizi che possono essere riutilizzati. Tuttavia, continuando ad effettuare un confronto con le griglie computazionali, essa risulta meno robusta rispetto a quella del Grid, poiché consente di effettuare operazioni privilegiate attraverso una scambio di informazioni via Web (sebbene su comunicazione sicura SSL), senza alcuni passaggi intermedi altrimenti necessari nelle griglie. Nelle griglie, progettate sotto l'assunzione che le risorse siano dinamiche ed eterogenee, la sicurezza è ingegnerizzata nell'infrastruttura fondamentale del Grid computing, avvelondosi di componenti dedicati quali il Grid Security Infrastracture (GSI) ed il Community Authorization Service (CAS). Nel Cloud, invece, l'infrastruttura si avvale di Security Services, attraverso un'interazione Web. Questa tipologia di servizi, permette agli utenti finali di accedere ai servizi disponibili, di gestire il loro ciclo di vita (in termini di creazione, modifica, eliminazione), di esplicare funzioni di autorizzazione nell'accesso agli stessi servizi, di garanzia della condifenzialità e dell'integrità delle informazioni scambiate (attraverso crittografia, firme digitali, controllo di accesso), di creazione e gestione di Tunnel Trusted Schema (TLS) tra il client ed il servizio, per mezzo di chiavi crittografiche. In letteratura [16], sono specificati sette fattori di rischio da tenere in considerazione da parte degli utenti che voglio immettere le loro applicazioni nel cloud:

- **Accesso privilegiato**: i dati sensibili che transitano al di fuori dell'ambiente enterprise devono essere accessibili e propagati solo ad utenti privilegiati;
- **Conformità alle norme**: un cliente deve poter verificare l'autorità e l'affidabilità delle infrastrutture cloud offerte dai provider;
- **Locazione dei dati**: poiché la posizione geografica dei dati all'interno del cloud, per suo proprio modello di deploying, è celata o al più fornita in modo approssimativo al consumer, è necessario che le modalità di archiviazione dei dati siano conformi alla giurisdizione di appartenenza e che siano affini a dei requisiti di privacy locali per il cliente;

- **Segregazione dei dati:** è necessario garantire che la visibilità dei dati di diversi consumer sia ristretta esclusivamente ai dati di loro appartenenza;
- **Recovery**: è essenziale che i provider garantiscano la presenza di meccanismi di replicazione e ripristino dei dati in caso di necessità;
- **Supporto nella ricerca**: da parte dei provider per facilitare le ricerche di servizi cloud,
- **Esistenza dei dati a lungo termine**: è necessario garantire che i dati siano sempre reperibili, anche in caso di grossi cambi strutturali ed aziendali da parte dei provider.

Da questi presupposti, si è arrivati a definire, nel corso del tempo, il concetto di Cloud Risk e di vulnerabilità su cloud computing. Poiché il cloud è sostanzialmente l'insieme di più tecnologie messe in un proprio modello, l'utilizzo del cloud computing ha comportato dei cambiamenti significativi nei fattori di vulnerabilità nel IT. In Fig. 21, sono illustrate alcune caratteristiche del cloud computing affette da specifiche vulnerabilità, espresse in termini percentuali. [17] La maggior parte di esse, riguarda le caratteristiche fondamentali discusse e ripetute nei precedenti paragrafi quali quelle inerenti alla condivisione delle risorse, piuttosto che quelle relative ai dati o all'accesso ai Cloud Service.
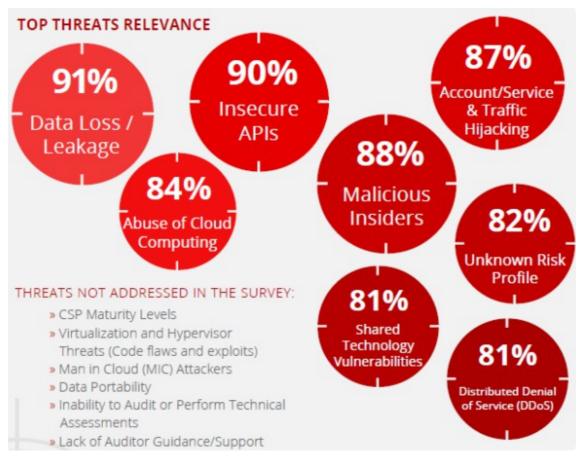
Fig. 21: Maggiori vulnerabilità nel cloud computing [17]

Dato un insieme di definizioni di vulnerabilità, e considerate le caratteristiche intrinseche nel cloud computing, è possibile definire delle vulnerabilità specifiche per il cloud [18]:

- Intrinseche o prevalenti in una componente fondamentale del paradigma o in qualche tecnologia caratteristica;
- Causate dall'impossibilità di erogare servizi di sicurezza;
- Prevalenti nello *state-of-the-art* del paradigma.

Vulnerabilità gravi sono quelle che coinvolgono l'hypervisor, successivamente quelle relative allo storage, alla risorse di rete e ai servizi cloud. Pertanto tutti gli sforzi sono incentrati nel controllo delle componenti fondamentali, sulla definizione di metriche di

sicurezza e schemi di certificazione che possano diminuire sensibilmente le probabilità di violazione del sistema, anche grazie ad ulteriori gradi di virtualizzazione, come recentemente avviene anche per i componenti di rete (es. VMware Nicira, OpenSwitch).

1.2.6 Altri aspetti

Oltre alle macro-divisioni concettuali all'interno del cloud considerate finora, vi sono alcuni ulteriori aspetti da discutere per avere un'esaustiva analisi e delle solide basi teoriche, al fine di poter affrontare successivamente gli aspetti prettamente pratici in merito adozione di un modello di cloud computing per le finalità di questo elaborato. Completano la tassonomia all'interno del cloud le definizioni di:

- **Load Balancing**: anche e soprattutto in un'architettura service-oriented, come quella in esame, è necessario prevedere dei meccanismi di *failover,* ovvero delle tecniche attraverso le quali delle discontinuità inattese nell'erogazione dei servizi in esecuzione, costantemente monitorati, vengono segnalate ad un *load balancer*, il quale interrompe la comunicazione con tali servizi non responsivi, evitando così inutili tentativi di connessione. Un load balancer è uno strumento che può essere utilizzato inoltre per razionalizzare il consumo delle risorse, nonché del consumo dell'energia, come avviene nel Green Cloud e nelle soluzioni energy-aware;
- **Interoperabilità**: risulta strategico avere la possibilità di spostare le proprie applicazioni su più cloud, oppure poter usare più infrastrutture cloud sul quale offrire i propri servizi. L'interoperabilità nel cloud è un concetto chiave, sul quale si stanno concentrando molti sforzi verso un processo di standardizzazione delle operazioni e delle metodologie di deploy, al fine di garantire alcuni dei concetti basilari analizzati precedentemente quali affidabilità e durata del servizio nel tempo, anche in caso di disastro;
- **Archiviazione dati scalabile**: è una proprietà più volte ripetuta nelle discussioni fatte nei precedenti paragrafi, e risulta in effetti uno dei maggiori motivi per il

quale si decide di migrare sul cloud. Il cloud offre una scalabilità orizzontale, ed attraverso il load balancer e le applicazioni, fornisce delle soluzioni. [15] La scalabilità verticale invece è relativa alle risorse usate. Pertanto è necessario progettare le applicazioni in funzione di tale scalabilità, al fine evitare che un fallimento nella scaling possa tradursi in un aumento del consumo delle risorse all'aumentare delle richieste.

2. SISTEMI DI CLOUD COMPUTING

“If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility [...] The computer utility could become the basis of a new and important industry” [19]

La frase citata come incipit di questo capitolo, di John Mc Carthy, è considerata dalla comunità scientifica come il concetto primordiale che ha dato luogo alla nascita e all'evoluzione della computazione distribuita in termini di cloud computing. Sebbene coniata in un contesto storico dove l'accesso alla potenza di calcolo fosse strettamente riservato a pochi soggetti, la visione di Mc Carthy trascende l'ormai capillare presenza di personal computer, poiché il cloud interpreta lo spirito delle sue parole, in funzione della possibilità di pubblico accesso a grandi risorse di calcolo, secondo un modello ormai standard. E l'industria di cui parla corrisponde pienamente al Information Technology. Anche il sogno degli hacker della prima generazione [20], divenuto ormai *ubiquitous computing*, coinvolge pienamente il cloud come mezzo per lo stesso nobile fine, ovvero che il computer possa essere utilizzato per migliorare la vita delle persone (concetto non propriamente scontato agli inizi degli anni 60).

Dal preambolo storico e filosofico, è possibile ora passare ad aspetti più pragmatici, tornando allo scopo di questo lavoro di tesi, ovvero la ricerca di una soluzione per lo sviluppo Web Enterprise nel cloud. Al fine di giungere a questo scopo, è necessario fornire una descrizione di quelli che sono i sistemi di cloud computing attualmente sul mercato, soffermandosi sulle diverse caratteristiche e funzionalità, che implementano i concetti teorici di base argomenti nel precedente capitolo.

Sebbene i sistemi cloud abbiano ormai raggiunto un livello di visibilità e di importanza tale da essere impiegati nella maggior parte di organizzazioni che svolgono attività di ricerca e di realtà aziendali impegnate nel IT, non è possibile fornire in modo esaustivo una lista di tutte le possibili soluzioni, senza considerare alcuni fattori di interesse e di classificazione. Un aiuto considerevole proviene dalla definizione dei modelli di servizio, all'interno dei quali collocare e classificare delle soluzioni esistenti, al fine di fornire un'ampia panoramica, in base alla quale determinare una scelta che soddisfi i requisiti richiesti per le proprie necessità. In questa overview, viene tralasciato il modello SaaS, poiché quest'ultimo viene offerti agli utenti finali appartenenti all'estremità superiore della piramide di Sheehan, dove applicazioni preconfigurate vengono semplicemente eseguite e non vi è alcuna conoscenza della tecnologia sottostante. È interessante invece, e inerente al lavoro di tesi, studiare i sistemi cloud in termini di soluzioni e tecnologie di infrastruttura (IaaS) e di provider di piattaforme (PaaS), cercando di estrapolare tutte le possibili combinazioni di queste ultime, con la massima interoperabilità.

2.1 Infrastracture-as-a-Service

Il modello di servizio IaaS, come visualizzabile in Fig. 3, occupa l'estremità inferiore della piramide di Sheehan, ed offre un controllo totale sull'infrastruttura server, non confinato a applicazioni o container, piuttosto che ad istanze restrittive. La Fig. 14 mostra le componenti di pertinenza del consumer in questo modello. Sono messe a sua disposizione risorse di calcolo, di archiviazione dati e di rete. Vi è inoltre, la possibilità di scegliere di effettuare il deploy di software arbitrario senza avere l'obbligo di gestire l'infrastruttura cloud sottostante, mantenendo tuttavia il controllo sul sistema operativo e le risorse virtuali, nonché ovviamente sui propri processi.

Fig. 22: Pertinenza di gestione nel modello di servizio IaaS [22]

Ciascun modello di infrastruttura presente sul mercato, si organizza sulla base di tre macro-unità fondamentali, illustrate in Fig. 23, che corrispondono a *Compute*, *Networking* e *Storage*.
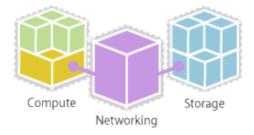
Fig. 23: Unità fondamentali in IaaS [50]

Queste unità sono virtualizzate, e soggette a *scheduler* di orchestrazione delle risorse all'interno dell'infrastruttura.

2.1.1 AWS: Amazon Web Services

Amazon è l'azienda precorritrice e leader indiscussa nel mercato del cloud computing, ed il suo successo nella divulgazione e nella commercializzazione della sua infrastruttura, è tale che essa stessa venga indicata come sinonimo di cloud.

Fig. 16: Amazon Web Services

Ha incentrato ingenti sforzi economici, sin dal 2005, per realizzare l'infrastruttura di larga scala, affidabile ed efficiente che sta alla base della più grande piattaforma di vendita online. Dal 2006, ha messo a disposizione questo know-how sotto forma di servizi cloud che prendono di AWS, attraverso i quali altre organizzazioni possono usufruire della propria infrastruttura, avendo un provider collaudato e totalmente distribuito. AWS è una piattaforma comprensiva di servizi cloud, ed oltre ad offrire le macro-unità fondamentali, offre un vasto insieme di funzionalità utili ai propri clienti; la Fig. 24 mostra tutte le offerte in AWS, organizzate in uno schema gerarchico a quattro livelli:
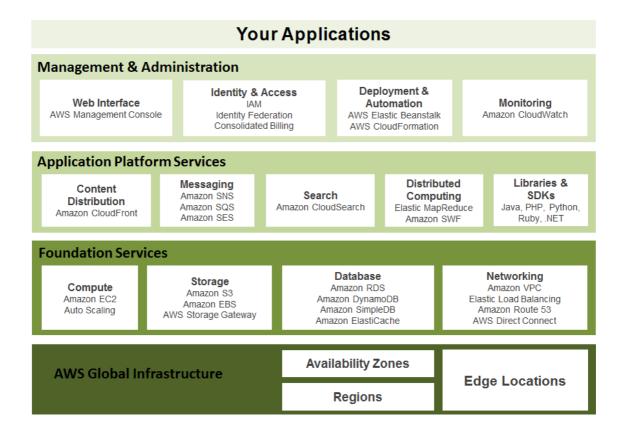
Fig. 24: Panoramica completa Amazon Web Services [23]

- **Management & Administration:** appartengono a questo livello i servizi di gestione Web e di monitoraggio delle componenti sottostanti;
- **Application Platform Services:** sono presenti servizi per l'interrogazione programmatica dei Web Service sottostanti, attraverso un insieme di API dedicate per i più comuni linguaggi di programmazione. È il PaaS per l'infrastruttura di Amazon;
- **Foundation Services:** è relativo ai tipici servizi cloud che definiscono un' infrastruttura, quali Compute, Storage, Networking, Persistence;
- **AWS Global Infrastructure**: è il Fabric di Amazon, suddivide l'infrastruttura a zone geografiche di pertinenza per gestire la distribuzione dei sistemi.

La totalità dei servizi cloud offerti in Amazon è molto ampia, esulando gli scopi di

questo lavoro di tesi; conviene bensì focalizzare l'attenzione sugli aspetti inerenti, ovvero i servizi di infrastruttura all'interno dell'unità di Foundation Services. Successivamente sarà possibile fare delle considerazioni su quelli di piattaforma

2.1.1.1 EC2 (Amazon Elastic Compute Cloud)

EC2 è un Web Service che fornisce nel cloud capacità computazionale personalizzabile, pensato e sviluppato in funzione del concetto di scalabilità della potenza di calcolo erogata. EC2 è il cuore del modello IaaS di Amazon, e permette, tramite un'intuitiva Console che invoca i servizi di computazione, di avviare delle istanze server in pochi minuti, fornendo ovviamente la possibilità di scalare la potenza di calcolo, in maniera semplice ed intuitiva. EC2 interpreta il concetto di elasticità nel cloud, poiché permette di aumentare o diminuire la capacità computazionale delle istanze in poco tempo e in base al carico di lavoro richiesto. Offre la possibilità di avviare migliaia di istanze simultaneamente, potendone mantenere il controllo anche in maniera programmatica, tramite le API del Web Service.

Amazon fornisce delle immagini preconfigurate e ottimizzate per l'infrastruttura di riferimento, generate per mezzo del sistema di template di AWS. È possibile creare la propria immagine personalizzata secondo un meccanismo di creazione di immagini Amazon Machine Image (AMI). Tali immagini, corrispondono essenzialmente in un particolare tipo di sistema operativo preconfigurato, utilizzato per creare macchine virtuali all'interno di EC2. È presente inoltre un repository dei contributi in termini di immagini AMI caricate dagli utenti di EC2 e condivise nel cloud, le quali seguono un modello di deployment in merito sicurezza specificato nella documentazione ufficiale. [25] È bene ricordare che un immagine AMI non corrisponde ad un'istanza in esecuzione, pertanto parlare di immagini condivise significa condividere lo stesso insieme di base di sistema, librerie e applicazioni contenuti nel sistema operativo personalizzato nell'immagine, ma non significa condividere la stesse risorse, che sono comunque isolate, come già visto nella server consolidation. Una volta scelta o caricata la AMI, è possibile gestire l'esecuzione delle istanze utilizzando i numerosi strumenti di

gestione disponibili dalla AWS Console o in modo programmatico attraverso le API del Web Service. Essendo un Web Service, lo sviluppatore può invocare i servizi tramite richieste HTTP, utilizzando REST o SOAP per la comunicazione, poiché vi è piena compatibilità con entrambi i protocolli. Dalla Fig. 25 è possibile visualizzare in maniera chiara questo processo.

Fig. 25: Invocazione delle API Web Service [26]

Il pagamento della quantità di computazione utilizzata è commisurata alla capacità realmente utilizzata. Si pagano dunque sono le risorse effettivamente consumate, in termini di istanza/ora o per quantità di dati trasferiti.

La computazione è definita in termini di EC2 Compute Unit (ECU), ed 1 ECU corrisponde a circa 1.2 Ghz di uno Xeon 2007. In base alle ECU, vengono definite dei tipi di istanza (flavor) che contengono sia le ECU, sia le altre risorse aggiuntive, le quali tutte insieme definiscono un piano base per la tariffazione di base delle istanze.

2.1.1.1.1 Auto Scaling

Come illustrato in Fig. 24, Auto Scaling è un componente appartenente all'unità di Compute, all'interno dei Foundation Services (servizi di infrastruttura) di Amazon. È stata progettata per aumentare le potenzialità di EC2, in termini di scalabilità. Difatti Auto Scaling permette scalare automaticamente le capacità computazionale in base a determinate condizioni definite. Con questo sistema, ad esempio, è possibile assicurare,

in modo del tutto automatico, che il numero di istanze di EC2 utilizzate possa incrementare in caso di picchi di richieste, oppure che possa diminuire in caso di mancanza di richieste, minimizzando del tutto i costi. Auto Scaling è controllato dall'unità di monitoraggio CloudWatch appartenente al dominio amministrativo dell'offerta cloud Amazon.

2.1.1.2 ELB (Load Balancing Service)

Appartenente sia all'unità di Compute, che a quella di Storage, ELB si fa carico di distribuire automaticamente il traffico entrante su più istanze EC2. Come osservato nel precedente capitolo, il Load Balancer è uno strumento che ottimizza il carico di dati da ridistribuire all'interno del modello; nel caso di Amazon, ELB riesce ad ottenere un elevato grado di tolleranza ai guasti, garantendo il corretto instradamento del traffico in entrata alle applicazioni. Le risorse di cui si fa carico il Load Balancer di Amazon sono le istanze, il quale rileva le problematiche di comunicazione relative alle istanze non responsive, ed instrada automaticamente il traffico verso altre istanze disponibile, fino alla risoluzione del guasto. Un Elastic Load Balancer in Amazon fa solitamente riferimento ad una singola Availability Zone, ma nei casi particolari può coinvolgere più zone.

2.1.1.3 VPC (Amazon Virtual Private Cloud)

Amazon VPC è la soluzione di networking privato virtuale all'interno dell'infrastruttura di Amazon. Permette di definire una rete virtuale privata, all'interno della quale è possibile utilizzare risorse AWS, in maniera del tutto isolata. È possibile definire una topologia di rete, alla stregua di quelle operate nei data center, avendo a disposizione la possibilità di controllare l'ambiente di networking virtuale, attraverso le classiche funzioni di indirizzamento IP, di creazione delle sottoreti e di opportuni percorsi di routing.

2.1.1.4 S3 (Simple Storage Service)

Amazon S3 è un Cloud Service per lo storage, progettato per agevolare lo sviluppo all'interno dell'offerta di cloud computing di Amazon. Fornisce un'interfaccia dedicata alla comunicazione tramite Web Service, attraverso la quale è possibile, in ogni instante, archiviare o recuperare qualsiasi mole di dati, distribuita automaticamente all'interno del cloud. Il contenitore di oggetti archiviati in S3 prende il nome di S3 Bucket [27], e le operazioni di storage seguono il processo illustrato in Fig. 19.
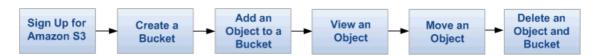
Fig. 26: Processo di gestione dei dati in S3 [27]

S3 è costruito su insieme essenziale di funzionalità:

- Scrittura, lettura ed eliminazione di oggetti nell'ordine di grandezza da 1 byte a 5 GB;
- Meccanismi di autenticazione forniti per garantire la sicurezza dei dati;
- Reperibilità ed integrità dei dati garantita.

2.1.1.5 ELB (Elastic Block Storage)

Amazon ELB è la soluzione di storage persistente per EC2. Fornisce dei volumi remoti di archiviazione dati persistenti, indipendenti dalle istanze, pertanto reperibili anche quando le stesse istanze non sono in esecuzione. Possono essere connesse a runtime, e sono utili per le applicazioni che richiedono la presenza di database, piuttosto che un particolare tipo di file system. Inoltre i volumi EBS offrono una maggiore durabilità

rispetto ai dati archiviati sulle istanze EC2, poiché replicate automaticamente sui backend. Vi è anche la possibilità, all'occorrenza, di creare delle snapshot dei volumi e di salvarle semplicemente in S3, secondo il processo esposto nel precedente paragrafo, ottenendo una replicazione automatica dei dati sulle Availability Zones di competenza.

2.1.1.6 DynamoDB

Amazon DynamoDB è il nuovo prodotto dell'offerta Database di Amazon, all'interno dei Foundation Services. Rappresenta l'alternativa NoSQL rispetto ad Amazon RDS (Relational Database Service), il Web Service di supporto ai più noti database relazionali. Sostituisce inoltre SimpleDB. DynamoDB coniuga le elevate prestazioni in termini di velocità e di capacità di predizione, con una collaudata scalabilità, risultando essere la scelta preferenziale per gli Al fine di mantenere le sue performance in termini di velocità e consistenza, DynamoDB ridistribuisce i dati e il traffico per le tabelle coinvolte su un numero sufficiente di server capace di gestire la capacità di richieste definite, insieme alla dimensione dello storage. Tutti i dati sono salvati su dischi a stato solido (SSD) e sono automaticamente replicati secondo il consueto modello delle Availability Zones.

2.1.1.7 Controllo e Gestione delle Risorse in AWS

L'insieme di web service offerti da Amazon rende l'offerta completa esaustiva per tutti gli utenti che vogliono essere presenti nel cloud attraverso la sua infrastruttura. È interessante citarne ulteriori due, in funzione del concetto di interoperabilità e di diverse implementazioni nel cloud:

2.1.1.7.1 AWS CloudFormation

AWS CloudFormation: è un servizio per la creazione e la gestione di risorse in AWS tramite un sistema di template. Per mezzo di un template, è possibile descrivere risorse AWS e definire delle dipendenze fra di esse o su parametri di runtime. Ad ogni template corrisponde una pila di risorse, di cui viene effettuato il deploy in base alle dipendenze e allo scheduling di CloudFormation.

2.1.1.7.2 AWS CloudWatch

Pensato per l'unità di Monitoring dell'offerta cloud, questo servizio dà la possibilità di monitorare le risorse cloud su AWS e le stesse applicazioni in esecuzione sui web service di Amazon. Sono fornite delle API per gli sviluppatori che intendono tenere traccia dei loro sistemi o delle loro applicazioni in modo programmatico, potendo realizzare dei software integrativi dell'offerta. Tramite il sistema di Alarm, è possibile prendere delle decisioni automatiche in base ad una metrica definita (nel template ad esempio), e questo risulta molto utile nel supporto della funzionalità di Autoscaling.
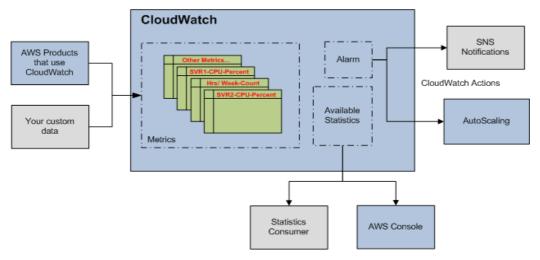
Fig. 27: Schema CloudWatch [25]

2.1.2 Eucalyptus

Eucalyptus, acronimo di Elastic Utility Computing Architecture Linking Your Programs To Useful Systems, è un provider che offre servizi di infrastruttura, sulla base del suo

software open source di infrastruttura, rilasciato sotto licenza GPLv3. La caratteristica fondamentale di Eucalyptus è quella di offrire, alle organizzazioni che lo adottano, la possibilità di ottenere dei cloud privati e ibridi, grazie alla sua piena compatibilità con Amazon AWS. Il cloud ibrido di Eucalyptus consiste, in sostanza, in un'infrastruttura privata, simile a quella di Amazon, con la possibilità di condividere il carico di richieste con il cloud pubblico di Amazon.
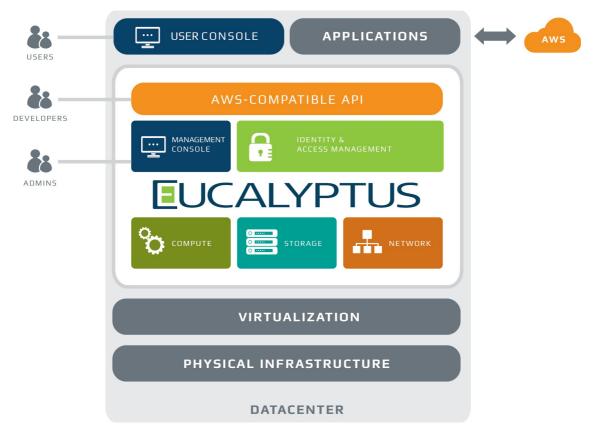
Fig. 28: Architettura di Eucalyptus [27]

Il nucleo di infrastruttura, visualizzabile in Fig. 28, è la base dell'architettura, che fa capo a risorse di computazione, di storage e di networking virtualizzate. Come in Amazon, vi sono delle immagini che descrivono un'istanza da eseguire. Queste sono denominate Eucalyptus Machine Instance (EMI), e contengono il file system di base. Tipicamente un'immagine contiene una distribuzione Linux ed oltre alle EMI, sono definite altre tipologie di immagine quali EKI ed ERI, rispettivamente per Kernel e

Ramdisk, che contengono i moduli del kernel necessari per il funzionamento della EMI.

L'architettura di Eucalyptus è altamente scalabile, grazie alla sua stessa predisposizione alla distribuzione delle risorse. Si suddivide logicamente in tre livelli:
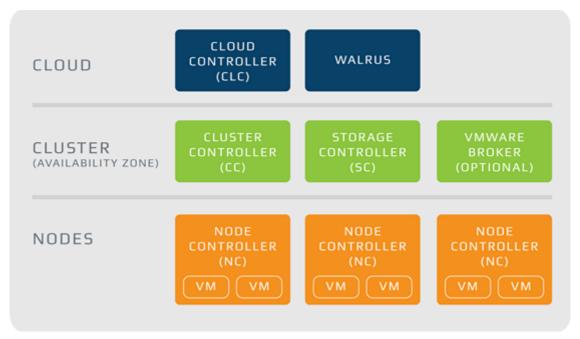
Fig. 29: Diagramma dei livelli di Eucalyptus [27]

- **Cloud**: il livello cloud è composto da due soli livelli, CLC e Walrus, utilizzati da molti utenti;
- **Cluster**: racchiude in sé il concetto di Availability Zone, discusso precedentemente, e contiene due controller di infrastruttura, ovvero di Cluster e di Storage, ed un Broker di rete virtuale;
- **Nodi**: il livello dei nodi è comprensivo di più componenti, ma ciascun componente supporta una porzione di utenti limitata, anche se le transazioni coinvolte sono grandi. [27]

Come si evince dalla Fig. 29, i livelli che costituiscono l'infrastruttura di Eucalyptus, contengono sei componenti distinti, i quali possono essere anche offerti su altre architetture.

2.1.2.1 CLC (Cloud Controller)

CLC è un software scritto in Java che offre interfacce di comunicazione Web Service, basate su SOAP, compatibili con Amazon EC2. Offre inoltre la possibilità di accedere ai servizi tramite un'interfaccia web, come avviene con la Console AWS. CLC effettua lo scheduling delle risorse ad alto livello, e rappresenta l'interfaccia di amministrazione per la gestione dell'infrastruttura cloud in Eucalyptus. Gestisce le risorse di computazione, di storage e di rete, ed prevede la comunicazione con tool appositi come euca2ools o programmi esterni per mezzo delle sue API.

Esiste un solo CLC per cloud e ciascun CLC gestisce ad alto livello:

- Autenticazione;
- Accounting utente;
- Report;
- Quota management

2.1.2.2 Walrus

Walrus è il Cloud Service relativo alla gestione dello storage, ed è l'equivalente di Amazon S3 per Eucalyptus. Offre un'archiviazione dati persistente per tutte le macchine virtuali all'interno del cloud di Eucalyptus, e può anche essere usato come una semplice soluzione SaaS per lo storage. Può contenere dati, snapshot di volumi o immagini EMI, ma può esistere un solo Walrus per cloud.

2.1.2.3 CC (Cluster Controller)

CC è il front-end per il livello Cluster all'interno della soluzione cloud di Eucalyptus, e comunica con i Controller di storage e di nodo, rispettivamente Storage Controller e Node Controller. I Cluster Controller supervisionano un insieme di istanze dalle quali estrapola delle informazioni da processare, ed effettua lo scheduling dell'esecuzione delle macchine virtuali su determinati Node Controller. Gestiscono inoltre le risorse di rete virtualizzate e contribuisce in maniera significativa al mantenimento dei SLA, diretti da CLC. Tutti i nodi serviti da un singolo Cluster Controller, devono essere nello stesso dominio di broadcast.

2.1.2.4 SC (Storage Controller)

SC implementa lo storage di rete, ed il suo compito è quello di creare dispositivi EBS persistenti . Può interfacciarsi con diversi sistemi di storage locali e remoti, quali ad esempio iSCSI ed NFS. Sebbene un volume EBS non possa essere condiviso da più istanze, è possibile, è possibile salvare delle snapshot dei volumi in sistemi di storage centralizzati come Walrus.

2.1.2.5 VMware Broker

VMware Broker è un componente opzionale che fornisce un interfaccia compatibile con AWS per ambienti virtualizzati con VMware. Viene eseguito fisicamente sul Cluster

Controller e si fa carico di gestire l'interoperabilità con eventuali host ESX esistenti, trasformando le EMI di Eucalyptus in dischi virtuali per VMware.

2.1.2.6 NC (Node Controller)

NC è un software eseguito su ogni nodo che contiene delle macchine virtuali di istanza gestisce gli endpoint della rete virtualizzata. Controlla il ciclo di vita delle istanze in esecuzione sul nodo ed interagisce con il sistema operativo e l'hypervisor per estrapolare i dati relativi alla disponibilità e all'utilizzazione delle risorse, che vengono successivamente inviati al Cluster Controller.

2.1.3 Joyent

Joyent è un innovativo provider europeo di IaaS, diretto competitor di Amazon, in forte crescita e con numerosi riconoscimenti, ultimo quello di "Technology of the Year" per il 2013 da InforWorld. [29] Nell'offerta Cloud di Joyent, è posta una particolare attenzione su prestazioni e scalabilità all'interno del modello di infrastruttura. Particolari accortezze sono adoperate nell'unità di networking, prevedendo politiche di caching automatico dei file system distribuiti, oltre a load balancing del traffico di rete. Tali misure permettono alla rete di essere conforme alla quantità di richieste alle applicazioni, in modo del tutto automatico, ottenendo addirittura, ove possibile, un risparmio ulteriore su ulteriori risorse virtuali.

La soluzione di cloud computing di Joyent è indirizzata quello nell'ottica di sopperire ai problemi di sovraccarico di rete AWS (outage). L'approccio è quello di disporre il livello di Applicazione e di Database nella stessa Zone, e spesso anche sulla stessa macchina virtuale, piuttosto che fare affidamento ad una rete esterna per connettere l'unità di computazione con quella relativa ai dati. [30] La Fig. 30 illustra la struttura logica dell'infrastruttura di Joyent.
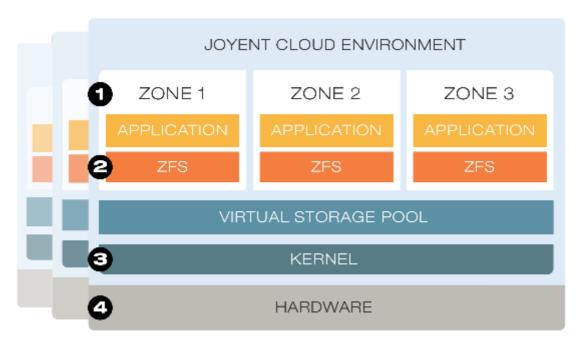
Fig. 30: Infrastruttura Cloud di Joeyent [30]

2.1.3.1 Zone

Joyent utilizza il concetto di Zone, familiare negli ambienti UNIX, e non a caso. Il cloud di Joyent nasce sulla base del sistema operativo di riferimento, SmartOS, UNIX based, che fa parte dell'insieme di immagini SmartMachine che il provider offre ai suoi utenti, secondo la prassi simile per le AMI ed EMI discussa precedentemente. Nei sistemi operativi, una Zone è definita come macchina virtuale *light-weight*. Permette di isolare un ambiente di esecuzione, infatti la sua progettazione è stata particolarmente incentrata

sulla sicurezza, aspetto non secondario in un contesto eterogeneo come quello del cloud computing. Il concetto di Zone all'interno cloud Joyent rappresenta un'ulteriore soluzione di sicurezza ed infatti il compito è quello di isolare l'ambiente di ciascuna applicazione.

2.1.3.2 ZFS

ZFS è il file system utilizzato da Solaris per la sua offerta UNIX, e viene adottato da Joyent per la gestione dello storage all'interno delle Zone. È stato adottato per la sua smisurata capacità, giacché è un file system a 128 bit, e perché fa uso di un modello transazionale ad oggetti di tipo *copy-on-write* . In un modello di questo tipo, tutti i puntatori ai blocchi all'interno del file system contengono un checksum a 256 bit del blocco puntato, di cui si fa uso per verificare la consistenza di ogni blocco in lettura. I blocchi che contengono dati attivi non vengono mai sovrascritti, piuttosto viene allocato un nuovo blocco per le modifiche dei dati, e ciascun blocco di metadati che lo referenzia viene letto, riallocato e scritto allo stesso modo. [31] Questo comportamento, favorisce il trattamento delle snapshot, che abbiamo visto essere prassi consueta all'interno delle soluzioni IaaS; quando ZFS scrive nuovi dati, mantiene i blocchi meno recenti, favorendo in modo strutturale delle snapshot del file system, denominate cloni.

Da tali considerazioni, si può dedurre facilmente il motivo dell'adozione di un file system come ZFS come componente strutturale dell'architettura di Joyent. ZFS gestisce all'interno di essa un pool di storage di dischi fisici virtualizzati, e quando uno di questi dischi fisici presenta degli errori, ZFS si fa carico di replicare automaticamente al nodo di computazione, in maniera del tutto trasparente e senza interruzioni per l'utente.

2.1.3.3 Kernel

Migliaia di Zone possono essere creata su un solo kernel. Il sistema operativo sul quale si basa l'infrastruttura Joyent, SmartOS, estende il sistema Illumos, un fork di Solaris, il quale rappresenta una continuità open source nella direzione di OpenSolaris, ormai abbandonato da Oracle. Sebbene predisposto al supporto di più hypervisor, SmartOS, e pertanto l'intera infrastruttura, adotta KVM come soluzione di riferimento per la virtualizzazione. Oltre alle elevate prestazioni offerte da questo hypervisor che hanno fatto ricadere la scelta su di esso, è l'ambiente di riferimento che lo rende particolarmente adatto ad una integrazione nella soluzione di Joyent, poiché le immagini KVM sono eseguite come un processo all'interno di una Zone, ottenendo una maggiore sicurezza grazie all'isolamento delle Zone, e traendo il vantaggio di un file system come ZFS.

2.1.3.4 Hardware

Le Zone virtualizzano le risorse fisiche, sulle quali convergono le applicazioni per mezzo del pool di storage virtuale. L'organizzazione in merito all'utilizzo delle risorse hardware viene definito all'interno dell'infrastruttura e delle Smart Machines.

2.1.3.5 SmartMachine

La SmartMachine è la tipologia di macchina virtuale di riferimento all'interno del Public Cloud di Joyent. Differiscono dalle tradizionali architetture di VM in primo luogo per la maggiore astrazione dall'hardware, poiché offrono alle applicazioni la possibilità di accedere ad un pool di risorse, rispetto al controllo di una risorsa fissa, come visualizzato in Fig. 23. [32]
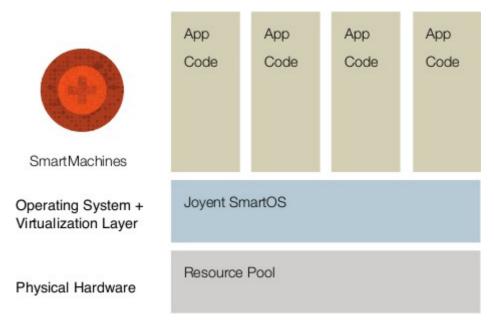
Fig. 31: Architettura di virtualizzazione delle SmartMachine [32]

Il cloud pubblico di Joyent si basa su SmartOS, ma supporta altri sistemi operativi nello stesso cloud, grazie all'integrazione di KVM. La distinzione nelle macchine virtuali disponibili è fra le SmartMachine, basate su SmartOS, ed le immagini KVM, ad esempio una distribuzione Linux o una versione di Windows. [33] La scelta ricade in base al tipo di applicazione necessario. Molti Web ed Application Server sono disponibili su piattaforma SmartOS, nonché Database come MySQL o ambienti di sviluppo come Node.js; pertanto gli sviluppatori possono avere il loro PaaS all'interno dell'offerta IaaS di Joyent, attraverso le SmartMachine. E qualora sia richiesta una particolare configurazione di sistema per un determinato applicativo, è possibile generare delle immagini personalizzata a partire dai template dei sistemi operativi supportati dal loro KVM.

2.1.3.6 SmartDataCenter

SmartDataCenter rappresenta un livello di astrazione ulteriore per l'intera offerta cloud, offrendo un unità di monitoraggio centralizzata ed esponendo delle API per gli

sviluppatori che vogliono gestire monitorare le proprie risorse in maniera programmatica. Rappresenta il livello di orchestrazione delle risorse all'interno del cloud di Joyent. Tramite SmartDataCenter, è possibile scalare in modo semplice un grande mole di dispositivi, grazie ad un sistema di agenti (Joyent Agent), i quali utilizzano un bus di messaggistica per il monitoraggio e la gestione delle risorse. Inoltre i protocolli di comunicazione utilizzati per i messaggi sono XMPP e AMQP, entrambi open source ed estendibili, garantendo una piena compatibilità implementativa.
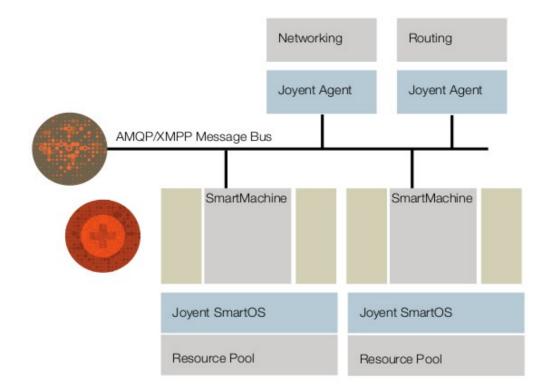
Fig. 32: Architettura di SmartDataCenter [32]

2.1.4 OpenNebula

OpenNebula è una solida soluzione open source di infrastruttura per la virtualizzazione di Data Center, perché progettato per essere integrato con qualsiasi tipo di soluzione di storage e di networking esistente. Lo schema visualizzato in Fig. 33, mostra le sue

componenti fondamentali:
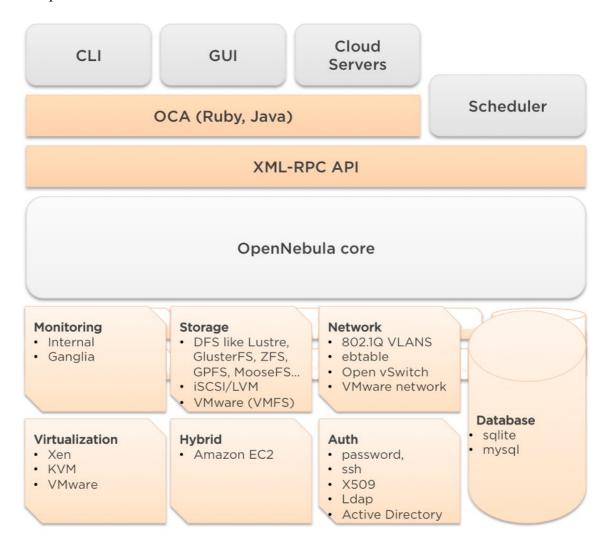
Fig. 33: Architettura di OpenNebula [34]

Partito inizialmente come progetto di ricerca nel 2005, attualmente è giunto alla versione 4 ed è diventato un'ottima soluzione per la creazione ed il mantenimento, anche a livello Enterprise, di Data Center virtualizzati e di cloud privati. [34] La struttura organizzativa di OpenNebula permette di esportare le proprie risorse di cloud privato su cloud pubblico, attraverso una serie di Cloud Service che espongono interfacce RESTful. I componenti coinvolti in questa conversione sono due Web

Service:

- **OpenNebula EC2 Query**: permette, all'interno di OpenNebula, di avviare e gestire macchine virtuali per mezzo delle API di Amazon EC2 Query. In questo modo, è possibile poter utilizzare qualsiasi strumento di EC2 Query per accedere il cloud privato di OpenNebula. Il Web Service è implementato sul livello OpenNebula Cloud API (OCA) che espone tutte le risorse di cloud privato;
- **OGF OCCI (Open Cloud Computing Interface)**: offre la possibilità di avviare e gestire macchine virtuali utilizzando le specifiche definite nelle API di OGF OCCI. OCCI è uno standard che comprende un insieme di specifiche promosse da una comunità di sviluppatori eterogenea, il cui scopo è l'interoperabilità e l'innovazione nel cloud. Le stesse sono raccolte ed approvate dal Open Grid Forum. [35]

La Fig. 33 illustra in maniera esaustiva ed autoesplicativa tutta l'offerta del cloud di OpenNebula. Sono supportati sia i più noti sistemi di virtualizzazione quali KVM, Xen, e VMware, sia le migliori soluzioni di storage remoto e virtualizzato quali GlusterFS, ZFS, VMFS. OpenNebula inoltre supporta un sistema di template per la descrizione e creazione di macchine virtuali, indipendenti da hypervisor e storage sottostante, È possibile definire un template e poi gestirlo con l'utility onetemplate oppure attraverso la dashboard di gestione Sunstone.

2.1.5 VMware vCloud

vCloud è la soluzione per il cloud computing di VMware, leader indiscusso dei sistemi di virtualizzazione di calcolo, di storage e di rete. In particolare, vCloud rappresenta un

insieme di Cloud Service comuni per provider che vogliono offrire servizi infrastruttura nel cloud, sia pubblico che privato. L'architettura è composta di un insieme di componenti della famiglia VMware, quali sistemi operativo, sistema di virtualizzazione e API per i servizi cloud, come illustrato in Fig. 34. In particolare:
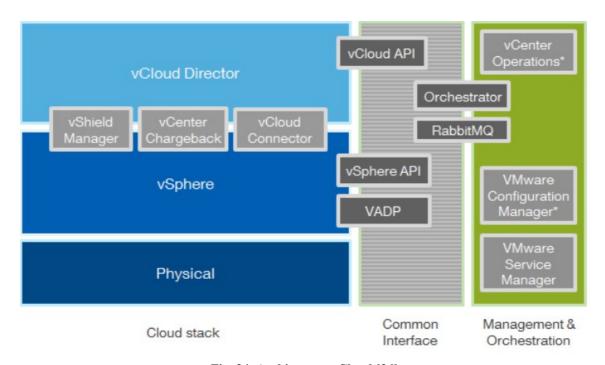
Fig. 34: Architettura vCloud [36]

- **vCloud Director**: livello software che astrae le risorse virtuali ed espone i servizi Cloud per mezzo delle vCloud API;
- **vSphere** : è il sistema operativo utilizzato per l'ambiente di virtualizzazione all'interno del vCloud. Contiene al suo interno ESX, l'hypervisor bare-metal della famiglia di virtualizzazione VMware;
- **vShield**: è un componente volto alla fornitura della sicurezza di rete perimetrale, per Data Center che utilizzano la soluzione VMware vShield Edge;
- **vCenter Chargeback**: contiene al suo interno una serie di strumenti di raccolta dei dati per il monitoraggio ed il report all'interno di vCloud;
- **vCenter Orchestrator**: è l'unità di orchestrazione dei task e delle risorse nel

cloud di VMware, organizzata in un'architettura a plugin.

La progettazione dell'infrastruttura architetturale di vCloud, prevede una separazione logica fra le risorse che devono essere allocate per le funzioni di management e di controllo, e le risorse dedicate alle richieste di lavoro da parte degli utenti. Questo al fine di ridurre l'overhead delle risorse allocate e per offrire una gestione consistente delle risorse di infrastruttura coinvolte. Tale suddivisione, si riflette rispettivamente in un Management Cluster, che contiene i componenti di gestione del vCloud come Director e Chargeback server, ed in più Resource Groups, che contiene le risorse utilizzate dagli utenti. Ciascun Resource Group inoltre, contiene più host ESX, sotto il controllo del Director. Infine, un Director può gestire le risorse di più Resource Group.

Le unità fondamentali dello stack cloud di computazione, di storage e di rete, sono gestite all'interno dei Management Cluster, e la Fig. 34 ne mostra l'organizzazione logica. Esse sono suddivise all'interno del Cluster in livelli di pertinenza:

- **Compute**: coinvolge CPU, memoria ed hypervisor. All'interno del cluster, sono presenti tre host, ed il sistema operativo vSphere si fa carico della configurazione e della dimensione delle risorse di computazione. Sono presenti degli strumenti che garantiscono la disponibilità per tutti i componenti volti alla gestione del cluster, quali il vSphere High Availability (HA), il Distribuited Resource Scheduling (DRS) ed il vSphere Fault Tolerance. Una menzione particolare va fatta per vSphere HA, il quale anziché definire il numero di interruzioni degli host che il cluster può tollerare, adotta delle policy di admission control basate su percentuali, del tipo N+1, ovvero prevede di ridistribuire il carico di lavoro in eccesso su tutti gli host;
- **Storage**: tutti gli host nel cluster accedono allo stesso insieme di datastore;
- **Network**: vi è una separazione logica del traffico di rete per ottimizzare il carico di lavoro e migliorare la sicurezza. Viene utilizzata la tecnologia VMware Virtual Distribuited Switching (VDS) per una maggiore semplificazione della gestione di rete attraverso la virtualizzazione del layer 2.
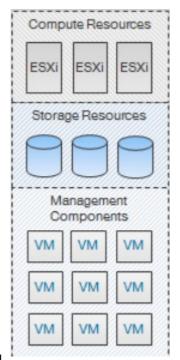
Fig. 35: Architettura vCloud [36]

2.1.5 Windows Azure Virtual Machines

Windows Azure è la soluzione di cloud computing di Microsoft, offerta in tre modelli di esecuzione, che corrispondono ai tre modelli di servizio del cloud. Windows Azure Virtual Machines è l'Infrastructure-as-a-Service che Microsoft offre come modello di Public Cloud, e fa riferimento in particolare alla creazione e gestione di macchine virtuali, secondo la struttura illustrata in Fig. 36.
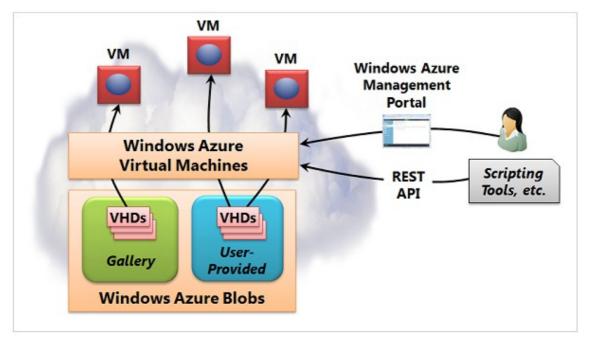
Fig. 36: Architettura IaaS in Azure [37]

I sistemi operativi contenuti nelle macchine virtuali sono salvati in un hard disk virtuale nel formato VHD, e rappresentano le immagini predisposte alla duplicazione da eseguire (come AMI, EMI, etc). Per generare le VHD, è necessario disporre di una versione di Windows Server, almeno dalla versione 2008 R2 per architetture a 64 bit. [37] I sistemi operativi disponibili sono quelli supportati dal hypervisor di Azure, Hyper-V, ed è possibile consultare la Windows Azure Virtual Machine Gallery o risorse esterne per ottenere delle immagini VHD di vari sistemi Windows e Linux, preimpostate per particolari configurazioni. [38]

È possibile creare macchine virtuali sia in maniera grafica ed intuitiva attraverso il Windows Azure Management Portal accessibile via web, sia in modo programmatico utilizzando le API dei Web Service REST offerti in Azure per la gestione delle istanze. Dopo aver lanciato un istanza, Azure automaticamente crea un disco di supporto per il salvataggio delle modifiche sulla macchina, che viene salvato nell'account di storage. Prevede inoltre lo spostamento automatico della macchina virtuale su altri server fisici in caso di problemi, sebbene consigli di eseguire più macchine virtuali nello stesso Availability Set, per raggiungere i SLA per la connettività esterna.
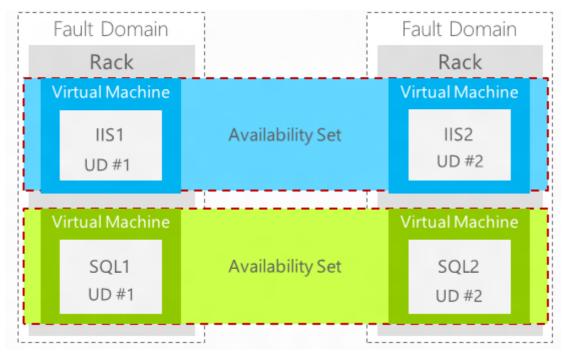
Fig. 37: Availability Set in Azure [39]

In Azure, la disponibilità delle applicazioni è garantita per mezzo di più macchine virtuali che appartengono ad un Availability Set, in combinazione con un Load Balancer. [39] Questi gruppi di macchine virtuali, sono uniti logicamente da alcuni domini di pertinenza:

- **Fault domains**: sono definiti per evitare che il guasto si verifichi in singoli punti critici come l'unità di alimentazione di un rack, oppure uno switch. Se vi sono più macchine virtuali connesse in un servizio cloud, si fa uso del Availabily Set per far sì che le stesse macchine appartengano a domini diversi.
- **Update domains:** sono domini utilizzati per garantire che le istanze non siano aggiornate tutte allo stesso istante. Dopo aver aggiunto più VM ad un Availability Set, Azure si fa carico di assegnarle a domini di update diversi. La Fig. 37 mostra un esempio dello scenario appena descritto.

2.1.6 OpenStack

Openstack è un progetto open source, sul quale convergono più organizzazioni, il quale scopo è di fornire un modello di servizio IaaS aperto, compatibile, altamente scalabile e

distribuito. Rilasciato sotto licenza Apache 2.0, nasce da uno sforzo comune di Rackspace, altro colosso del Public Cloud, e della NASA, ed è attualmente la soluzione di riferimento di più soluzioni di cloud computing (Rackspace, Ubuntu MaaS, Red Hat RDO). Scritto in Python, presenta un architettura modulare, i quali componenti corrispondono a delle unità specifiche dello stack cloud. Una descrizione dettagliata fornita e discussa nel capitolo successivo.

2.2 Platform-as-a-Service

Nella sua definizione più generale, il modello di servizio PaaS offre una piattaforma su cloud computing che espone su Internet l'infrastruttura ed i servizi necessari allo sviluppo, alla preparazione e all'esecuzione di applicazioni e altri servizi, come ad esempio portali. [40] Il Platform-as-a-Service, collocato al livello intermedio nella piramide di Sheehan, rappresenta la scelta ideale per le organizzazioni che non hanno particolari riserve nella scelta dell'infrastruttura, ma piuttosto indirizzano la loro ricerca verso una piattaforma di sviluppo dove effettuare il deploy delle loro applicazioni, pronte ad essere distribuite e scalate automaticamente, secondo la consuetudine del cloud computing. La Fig. 38, mostra, alla stregua della Fig. 22, le componenti di pertinenza dei due attori principali nel cloud, consumer e provider. Nel caso del PaaS, il grado di libertà del consumer si riduce al livello delle applicazioni, il quale, dopo averle sviluppate e pacchettizzate per la il runtime più opportuno, può caricarle all'interno della piattaforma ed eseguirle, senza doversi curare delle operazioni di manutenzione di server, DB, Container e di sistema, appartenenti al livello sottostante.
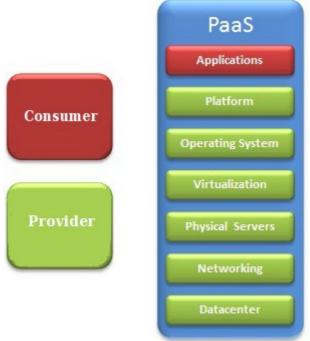
Fig. 38: Pertinenza di gestione nel modello di servizio PaaS [22]

Poiché lo scopo del lavoro di questa tesi è quello di ricercare delle soluzioni nel cloud per lo sviluppo Web Enteprise, viene fornita di seguito un'esaustiva panoramica delle soluzioni attualmente sul mercato, con particolare attenzione sia sulle tecnologie coinvolte per i container delle applicazioni, sia sulla possibilità di estensioni ed adattamenti delle platform su diversi ambienti di infrastruttura eterogenei.

2.2.1 Google App Engine

Google App Engine (GAE) è il modello di servizio PaaS offerto sull'infrastruttura di Google Cloud. Presente sin dal 2008, è una delle prime piattaforme di sviluppo web rivolte agli sviluppatori alla ricerca di un modello di scaling automatico delle loro applicazioni, ai quali è permesso di usufruire di tale servizio in maniera del tutto gratuita, nei limiti di un pacchetto standard di risorse utilizzate. App Engine ha rappresentato la soluzione di riferimento per gli sviluppatori Java che necessitavano

effettuare il deploy delle Web Application (Servlet, JSP) in modo semplice ed economico, poiché i servizi di hosting con supporto a tecnologie Java sono sempre molto cari e poco propensi a personalizzazioni ed aggiornamenti degli ambienti e dei container. Dal concetto di Common Gateway Interface (CGI) pubblico, si è passati nel corso del tempo, grazie alla diffusione del linguaggio Java, a quello di Servlet pubblica (es: Myjavaserver.com) per poi arrivare al generico, multi-purpose concetto di PaaS, che App Engine ha interpretato sin dagli albori. Attualmente GAE supporta più linguaggi di programmazione, quali Java, Python, Go, e PHP in termini di ambienti di sviluppo runtime. Per altri linguaggi molto diffusi in questi contesti come Ruby o Javascript, è invece previsto il supporto all'interno del runtime di Java, limitato a funzionalità di scripting.

Come già accennato, GAE è gratuito fino ad una certa soglia di risorse utilizzate. L'insieme basico di risorse di computazione, storage e di rete previsto all'interno del pacchetto di servizi gratuiti della piattaforma, prevede 1 GB di archiviazione dati, e risorse di calcolo e di rete sufficienti per gestire 5 milioni di visite (richieste web ai servizi) al mese. [41] Superata tale soglia, sono tariffate le risorse di storage e di rete nell'ordine del GB, e vi è la possibilità di controllare il processo di consumo delle risorse dalla Console web di amministrazione, in base alle necessità richieste dalla mole di utenti esterni che interagiscono con le applicazioni sul web.

2.2.1.1 Architettura di App Engine

App Engine è essenzialmente una piattaforma costruita su runtime Python, che offre hosting, storage, e servizi di rete ad alta velocità per applicazioni web, grazie all'infrastruttura di Google sottostante (Google Cloud). Come illustrato in Fig. 39, la comunicazione con l'esterno avviene per mezzo di semplici richieste HTTP, che attraverso il Load Balancer della piattaforma, vengono smistate secondo la maniera più opportuna all'interno dei cluster contenenti i nodi di computazione.
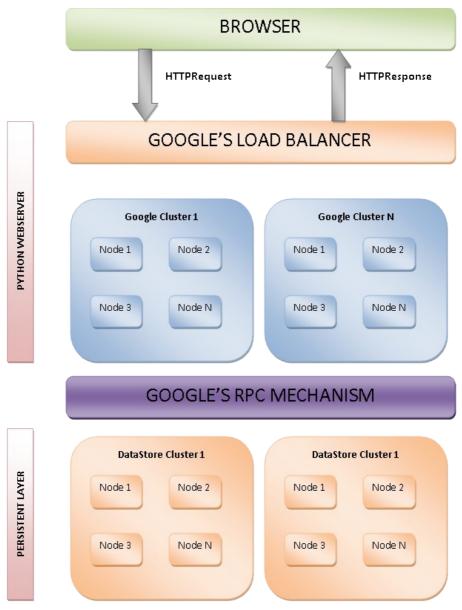
Fig. 39: Architettura Google App Engine

La comunicazione con i nodi di storage avviene per mezzo del meccanismo RPC, che unisce logicamente i due livelli di runtime e persistenza. Tale meccanismo è previsto anche all'interno del modello di sviluppo delle applicazioni. Grazie alla sua struttura ed organizzazione logica, GAE offre agli utenti finali tre distinte funzionalità relative al cloud computing:

- Una Platform-as-a-Service per gli sviluppatori che intendono esporre le loro

applicazioni web su Internet, in un sistema interamente costruito sul concetto di scalabilità e distribuzione del carico di lavoro, affidabile e sicuro;

- Le applicazioni create vengono offerte come un SaaS, utilizzate attraverso i browser degli utenti finali;
- La possibilità di integrare servizi web di terze parti da altre piattaforme o servizi.

2.2.1.2 Sviluppo

Lo sviluppatore che intende usufruire dei servizi di platform su GAE per l'hosting delle proprie applicazioni, ha a disposizione anche un processo di sviluppo semplificato e del tutto integrato con la piattaforma. Viene fornito infatti il Google App Engine SDK, che contiene serie di strumenti, interfacce ed API che permettono di controllare tutto il ciclo di vita dell'applicazione, dalla scrittura del codice, al deploy e al mantenimento della stessa. Sono disponibili inoltre, per i maggiori IDE quali Eclipse o Netbeans, dei plugin per App Engine SDK che permettono di lanciare le applicazioni sul cloud direttamente dal IDE utilizzato sulla workstation locale, con strumenti di monitoraggio e debug annessi. La Fig. 40 illustra il ciclo di vita di un'applicazione GAE.

Le fasi di *Build*, *Test* e *Deploy* appartengono al processo di sviluppo locale, mentre *Manage* ed *Upgrade* sono effettuate per mezzo della GAE Administrator Console, direttamente sul web. La Console è inoltre il luogo dove risiedono tutte le impostazioni delle applicazioni pubbliche offerte tramite App Engine (hostname, operazioni su datastore, permessi utente, statistiche).
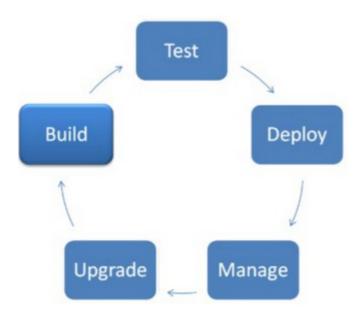
Fig. 40: Ciclo di vita delle applicazioni in GAE [41]

2.2.1.3 Storage

Poiché il modello PaaS è offerto solitamente su un IaaS, si è vincolati alle scelte d'infrastruttura dei provider, usufruendo di un grado di libertà relativo sono ai container delle applicazioni. Gli sviluppatori hanno la scelta di poter usare tre soluzioni diverse:

- **App Engine Datastore**: è il modello per lo storage di riferimento in GAE, ed è anche quello compreso nella soluzione gratuita. È un datastore distribuito NoSQL basato su Bigtable, *schemaless,* con supporto di transazioni atomiche (ACID) e con un motore di query. La struttura delle entità che devono essere rese persistenti viene definita all'interno dell'applicazione, gestibili grazie ad un insieme di interfacce presenti nei runtime, come ad esempio Java JDO/JPA. Le transazioni invece sono implementate nella rete distribuita del datastore mediante il concetto di *entity group*; una transazione può interagire con le entità all'intero dello stesso gruppo, e difatti per ragioni di efficienza, le entità

appartenenti allo stesso gruppo vengono salvate insieme.

- **Google Cloud SQL**: è la soluzione appartenente ai servizi di infrastruttura di Google Cloud, che permette di usufruire di DB relazionali come MySQL e PostgreSQL. Viene erogata anche agli utenti in GAE, che possono svincolarsi dalle ristrettezze e dai vincoli imposti dal Datastore, pagandone i servizi secondo il classico modello di economia di scala di consumo delle risorse;
- **Google Cloud Storage**: è il Cloud Service per lo storage di Google Cloud, per l'accesso dei dati nell'infrastruttura di Google attraverso web service RESTful.

Google App Engine Datastore rimpiazza il precedente modello di archiviazione dati all'interno della piattaforma, ed evolve nella direzione del cloud, ponendosi come alternativa ad Amazon DynamoDB, altra soluzione NoSQL discussa nel paragrafo 2.1.1.7 . Non essendo un database relazionale, le interrogazioni al database non sono eseguite per mezzo di SQL, ma attraverso delle query preparate all'interno del datastore. Ciascuna entity ha più proprietà, ed è identificata da un *kind* ovvero un tipo che la categorizza per le finalità della query, e da una *key* ovvero una chiave che insieme al kind identifica univocamente un entity. Esiste un'altro metodo per le interrogazioni che sono effettuate per mezzo di un linguaggio SQL-like di nome GQL; quest'ultimo permette di ottenere entity e key. La Fig. 32 illustra schematicamente questi concetti.

La scelta di una soluzione NoSQL, su base Bigtable, fornisce la garanzia di una scalabilità immensa e di una maggiore velocità di interrogazione rispetto ai più comuni RDBMS. [42] App Engine ha introdotto recentemente anche un nuovo sistema di replicazione del datastore denominato High Replication Datastore (HRD), basato sull'algoritmo Paxos [43], il quale permette il salvataggio sincrono dei dati su più datacenter. È possibile eseguire query con risultati conformi a modelli di consistenza stretta, utilizzando query ancestrali [44] limitate ad un singolo entity group, che come già detto rappresenta un insieme per le transazioni, ma anche un'unità di consistenza.

Un ultimo, ma non meno importante, accenno va fatto per Memcache, un servizio di memoria cache distribuito utilizzabile dalle applicazioni per aumentarne prestazioni e scalabilità. È rimandata agli sviluppatori il corretto uso di questa funzionalità poiché

sebbene favorisca una maggiore velocità nelle operazioni di lettura e scrittura, non è tuttavia garantita la presenza del dato in caso di grossi carichi di memoria, pertanto bisogna valutare bene quando e come usare la cache per i dati. Memcache supporta massimo 1MB ed è possibile definire un expiration time, attraverso il quale si garantisce che il dato non esisterà oltre, ma potrebbe anche non essere reperibile prima.
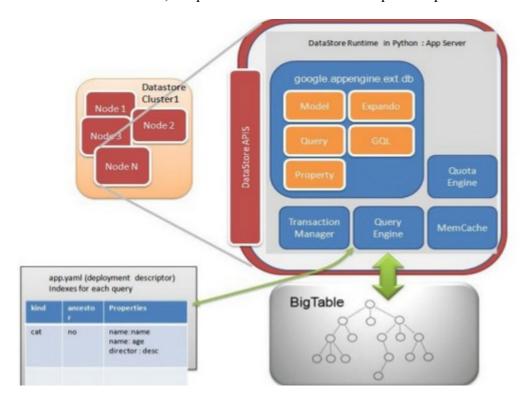
Fig. 41: Architettura Google App Engine [41]

2.2.2 Windows Azure Cloud Services

Cloud Service è il PaaS di Microsoft all'interno dell'offerta cloud di Windows Azure. La platform è offerta sul modello di infrastruttura visto nel paragrafo 2.1.5, rivolta agli sviluppatori che usufruiscono di un ambiente per le loro applicazioni, senza doversi curare del tipo di macchina virtuale da utilizzare. Le macchine virtuali vengono predisposte e organizzate in base al tipo di istanza:

- **Web Role**: istanze con una versione di Windows Server e Internet Information Services (IIS), il web server di Microsoft;
- **Worker Role**: istanze con una versione di Windows Server
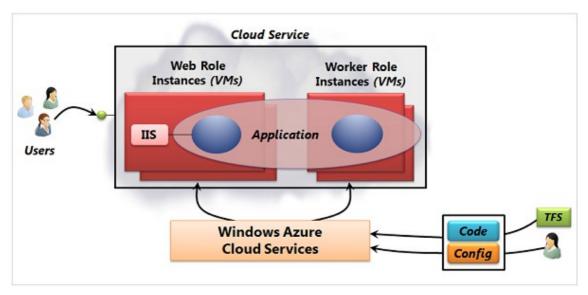
Fig. 42: Struttura Windows Azure Cloud Services

Come mostrato in Fig. 42, l'applicazione risiede all'interno delle due unità. Questo dominio logico rappresenta l'insieme degli scenari possibili all'interno della platform poiché nonostante un'applicazione possa usare un Web Role per le operazioni da svolgere a seguito di richieste HTTP dall'esterno, essa può smistare le stesse operazioni successivamente su un Worker Role, attraverso un canale di comunicazione come il Service Bus. Il numero di macchine virtuali da utilizzare per ogni Role può essere definito mediante un file di configurazione, ma è la platform che le crea

autonomamente. Lo scaling infatti avviene in termini di aumento delle istanze dei Role o viceversa di diminuzione delle stesse.

Cloud Services è l'ambiente naturale per lo sviluppo di applicazioni .NET sul cloud, ma sono supportati nella platform anche altri framrwork come Java, PHP e Python attraverso alcuni passaggi aggiuntivi. Il processo di sviluppo prevede la scrittura del codice con l'IDE di riferimento VisualStudio, insieme al Windows Azure SDK. Il codice compilato e pronto per il deploy del cloud viene caricato direttamente nella platform, risiedendo inizialmente in un area di staging. Successivamente l'applicazione viene messa in produzione tramite il Windows Azure Management Portal. Ciascun Role contiene al suo interno un agente, utilizzando per il monitoraggio delle applicazioni e delle macchine virtuali appartenenti al Role, che è in grado di rilevare problemi di esecuzione ed eventualmente di eseguire delle nuove istanze per entrambe. Cloud Services non prevede la persistenza all'interno delle macchine virtuali utilizzate dalle applicazioni, gli sviluppatori devono perciò rimediare mediante DB SQL, blob, o dischi di archiviazione dati esterni.

2.2.2.1 Storage

Cloud Services fornisce varie soluzioni per lo storage delle applicazioni, fra le quali i Blob e i database SQL. I blob vengono usati per file di testo e file binari non strutturati, e hanno una grande capacità, nell'ordine dei 100 TB per account Storage. Sono accessibili tramite servizi REST, attraverso i quali poter effettuare le operazioni su di essi, e vengono a loro volta suddivisi in Block blob e Page blob.

2.2.3 Heroku

Heroku è un'altra soluzione di PaaS offerta al pubblico con supporto multiplo di

ambienti di programmazione nonché di funzionalità comuni alle piattaforme cloud quali database NoSQL e SQL, Memcache ed un sistema modulare per l'integrazione di altre feature. Supporta i runtime più comuni ovvero Ruby, Java, Python, Node.js ma anche Scala e Clojure, e ha avuto molto successo presso gli sviluppatori di applicazioni web su base Ruby on Rails, tanto da divenirne la piattaforma di riferimento.
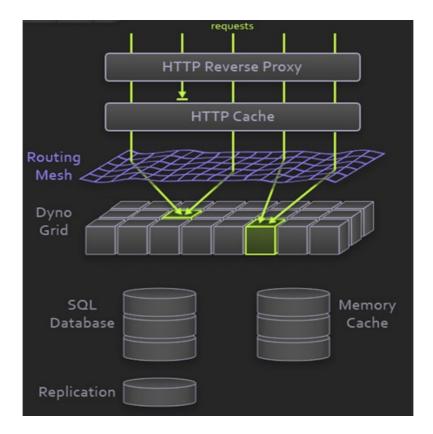
Fig. 43:

Architettura di Heroku [45]

L'architettura di Heroku, illustrata in Fig. 43, si fonda sul concetto di Dyno. Un Dyno è un contenitore che contiene l'ambiente di sviluppo per le applicazioni degli utenti, insieme alle applicazioni stesse. L'architettura prevede una griglia di contenitori Dyno, affinché le applicazioni possano essere distribuite all'interno di questa griglia in più Dyno, in funzione dello scaling delle applicazioni stesse effettuato automaticamente da Heroku. Le richieste HTTP dall'esterno vengono trattate da un Load Balancer ed immesse nella rete interna attraverso la funzione di HTTP routing [45] svolta nella piattaforma. Ciascun Dyno è conforme e preconfigurato in funzione delle proprietà

elasticità ed autoscaling, distribuzione e gestione della ridondanza e di isolamento nel proprio container subvirtualizzato. [46] La presenza di un Dyno Manager permette di avere un controllo ad alto livello sulle applicazioni, con riavvii e sospensioni automatiche in caso di necessità.

Heroku basa ed offre i suoi degli stack, ovvero un insieme comprensivo di sistema operativo, ambienti di runtime e librerie. La versione attuale è denominata Cedar, ed è strutturata in modo da fornire un ambiente di sviluppo per la creazione di applicazioni web SaaS conforme alla specifica Twelve-factor. [47]

2.2.3.1 Storage

La persistenza è gestita come add-on all'interno della piattaforma, fornendo PostgreSQL come soluzione preferita, ma anche MySQL. È possibile anche collegare storage esterni con Amazon S3 o Amazon RDS, mentre MongoDB si ottiene attraverso il concetto di DB-as-a-Service, ottenendo il proprio database sulla platform per mezzo delle istanze Amazon EC2.

2.2.4 CloudFoundry

CloudFoundry di differenzia dalle precedenti platform analizzate perché open source e pertanto non strettamente legate ad un determinato provider di servizi di piattaforma nel cloud. Rilasciata sotto licenza Apache 2.0 da VMware, è principalmente sviluppata in Ruby ed è offerta come modello di deploying pubblico sull'infrastruttura di VMware, godendo dei servizi di virtualizzazione di vSphere, analizzati nel capitolo 2.1.5. Supporta nella versione pubblica i runtime più comuni come Java, Python, PHP, Ruby, Node.js, ma anche Scala ed Erlang [48]; tuttavia, essendo open source ed estendibile, non ci sono limiti teorici alle possibili ulteriori configurazioni. La Fig. 44 mostra l'architettura complessiva di CloudFoundry, analizzata di seguito.
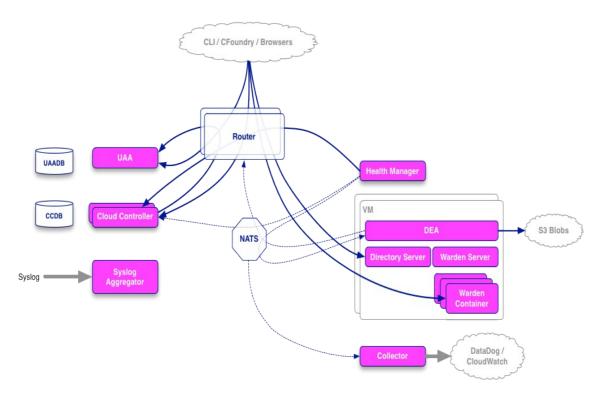
Fig. 44: Architettura di CloudFoundry [48]

Come si evince dal grafico, l'interazione con l'esterno è effettuato o per mezzo dei browser oppure della Console CLI, e smistate all'interno della piattaforma attraverso attraverso routing, tipicamente verso il CloudController o verso i nodi DEA, descritti insieme alle altre componenti di seguito:

- **Cloud Controller**: è un servizio che espone delle API REST per accedere al sistema. Mantiene un database con tutte le informazioni sulle applicazioni e i servizi all'interno della piattaforma (CCDB);
- **UAA (User Account and Authentication Server)**: è il sistema di autenticazione all'interno della piattaforma per le applicazioni eseguite per conto degli utenti all'interno di CloudFoundry. È un provider OAuth2, e può essere usato anche come Single-Sign-On (SSO) ad approccio centralizzato per l'autenticazione una tantum;
- **Execution Agent (DEA)**: è un servizio che si fa carico di gestire il ciclo di vita

della applicazioni. Tiene traccia delle applicazioni in esecuzione e notifica aggiornamenti in merito al loro stato tramite il NATS. Ciascun DEA fa riferimento ad unico Stack;

- **Servizio di messaggistica (NATS)**: è un sistema distribuito di tipo *publish-subscribe* per lo smistamento di code di messaggi all'interno della piattaforma;
- **Stack**: come già visto per le altre piattaforme, uno stack corrisponde ad un insieme comprensivo di sistema operativo e ambienti di runtime con alcune configurazioni. Attualmente CloudFoundry supporta un solo stack, ovvero lucid64, corrispondente alla versione di Ubuntu Linux 10.04 [49]
- **Warden**: è un ambiente per la gestione dell'isolamento delle risorse, che fornisce delle API per la gestione di tali ambienti, che prendono il nome di container. Un container è un unità di risorse che possono essere definite in termini di potenza calcolo, memoria, e di rete;
- **Servizi**: CloudFoundry prevede una serie di servizi a supporto della piattaforma, Gli sviluppatori che intendono supportare i loro servizi o quelli di terze parti possono implementare un servizio custom grazie alle API fornite dalla platform, ed eseguirli nella loro soluzione di PaaS privato.

Il punto forte di Cloud Foundry è la possibilità di estendere i servizi di piattaforma, diventando dei provider locali, e BOSH, un potente framework per il deploy e la gestione del ciclo di vita di servizi distribuiti su ampia scala. Inoltre BOSH è *general-purpose*, e permette l'integrazione con modelli di IaaS quali, ad esempio Openstack ed EC2. Come visualizzato in Fig. 45, ciascuna macchina virtuale contiene degli agenti, e per mezzo dell'interazione fra Director e agenti, vengono eseguite delle operazioni (Jobs) sull'istanza, utilizzando il sistema di messaggistica analizzato precedentemente.
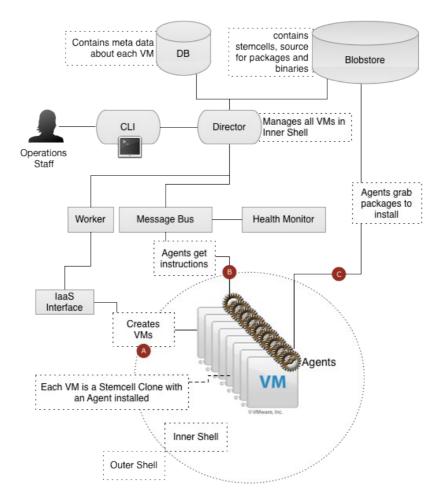
Fig. 45: Schema funzionamento BOSH [50]

2.2.5 Openshift

Openshift è un altro modello di servizio PaaS open source, realizzato da Red Hat, che dà la possibilità di offrire modelli di cloud pubblico e privato, anche per platform. La versione open source è denominata Openshift Origin, e è rilasciata sotto licenza Apache 2.0. Red Hat fornisce la Enterprise di Openshift come servizio cloud privato per organizzazioni che necessitano di un supporto personalizzato qualificato su base RHEL 6.0, ma è ovviamente possibile personalizzare la versione Origin sulle derivate open source dela sistema Red Hat, quali Fedora e CentOS. Anch'esso supporta i più comuni runtime ed una discussione approfondita a tale proposito verrà fatta nel capitolo 3.

2.3 DeltaCloud

Uno degli obiettivi di questo elaborato è quello di ricercare, a fronte di un'esaustiva panoramica sulle più interessanti proposte di modelli di servizio nel cloud computing, delle possibili combinazioni di questi modelli che possano fornire delle soluzioni in ambito IT, il più possibile astratte ed estendibili. La ricerca di interoperabilità nei cloud è la nuova frontiera su cui convergono più sforzi, ed esiste appositamente un progetto della celebre Apache Software Foundation, dal nome DeltaCloud [84], che va verso questa direzione. DeltaCloud offre un'astrazione del modello IaaS mediante delle API RESTful, permettendo allo stesso codice di essere valido su più provider. Raggiunge l'interoperabilità dei cloud a livello infrastrutturale, sul quale possono convergere le platform aperte. Fra i provider supportati vi sono quasi tutti quelle elencati, quali Openstack, OpenNebula, Amazon EC2, e la lista è in continua espansione. È possibile con delle semplici richieste HTTP, come illustrato in Fig. 46, lanciare istanze, monitorarle, creare bucket, snapshot, piuttosto che configurare load balancer e firewall; è anche disponibile un interfaccia web, scritta in jQuery Mobile, ottimizzata per tablet e cellulari, per una gestione più intuitiva delle opzioni offerte da DeltaCloud.

Fig. 46: Schema logico DeltaCloud [84]

3. PaaS over IaaS tramite Openshift ed Openstack

La seconda rivoluzione industriale non si presenta come la prima con immagini schiaccianti quali presse di laminatoi o colate d'acciaio, ma come i bit d'un flusso d'informazione che corre sui circuiti sotto forma d'impulsi elettronici.

Le macchine di ferro ci sono sempre, ma obbediscono ai bit senza peso. [51]

L'incipit di questo capitolo, è tratta da "Le Lezioni Americane" di Italo Calvino, un promemoria per il millennio corrente con sei concetti chiave che l'autore sceglie come concetti assimilati nel vecchio millennio e da portare nel nuovo, contestualizzandoli alla letteratura. La frase che ho scelto è tratta dal capitolo sulla Leggerezza, che l'autore espone come un valore, lieve come la luna di Leopardi, percezione di ciò che è infinitamente minuto e mobile come la realtà per Lucrezio. Nelle lezioni di Calvino, vi sono a mio avviso i punti chiave dell'evoluzione della società post-industriale o meglio nota come società dell'informazione, che va sempre più nella direzione del paradigma del cloud computing:

- Leggerezza;
- Rapidità;
- Esattezza;
- Visibilità;
- Molteplicità;
- Coerenza.

Rispettivamente, il calcolo "sulle nuvole", la velocità delle operazioni, il controllo sulle

operazioni, la reperibilità dei dati, la replicazione e la consistenza sono tutte prerogative e proprietà fondamentali nel paradigma del cloud computing, ed evidentemente la letteratura non è poi così lontana dalle necessità pratiche di altri ambiti. Sarebbe possibile estendere questi concetti chiave al caso del cloud, aggiungendo e contestualizzando il concetto di interoperabilità. Questo paradigma, nato dalla ricerca, è divenuto nel corso del tempo qualcosa di strettamente legato a pochi provider elitari, capaci di possedere risorse di calcolo smisurate da elargire in maniera condivisa e razionale secondo i modelli analizzati. Oggi il cloud invece è anche un metodo per l'ottimizzazione delle proprie risorse, e, grazie agli sforzi comuni incentrati verso la standardizzazione e l'interoperabilità, un modo per le organizzazioni di ridistribuire i loro carichi di lavoro computazionale, di archiviazione e di rete, su più data center, più provider, più servizi, in modo del tutto trasparente e secondo il consueto concetto di economia di scala.

Come mostrato in Fig. 22, il grado di libertà nelle PaaS è ristretto solo al dominio delle applicazioni, affinché i consumer non si debbano fare carico della gestione delle risorse di infrastruttura sottostanti. Questo è il classico scenario che il mercato propone, proponendo il livello IaaS per le soluzioni personalizzate, ma di recente si sono concentrati alcuni contributi significativi nella direzione della scelta dei servizi di platform da erogare, e le community molto attive in questo senso. Di seguito vengono esposti due progetti open source di larga portata che si adattano benissimo a queste considerazioni, dove sviluppatori ed utenti finali convergono i loro sforzi per ottenere dei prodotti free-as-in-freedom [52]

3.1 Openstack

Come accennato nel paragrafo 2.1.6, Openstack è un progetto il cui scopo è la realizzazione di un modello di servizio IaaS open source, che possa offrire modelli di cloud pubblico e privato, in modo semplice, estremamente scalabile e modulare. Difatti, la *mission* esplicitamente dichiarata sul sito ufficiale è quella di una piattaforma ubiqua di cloud computing open source, che interpreti tali prerogative. [53] Il cloud pubblico è offerto per mezzo di provider che utilizzando Openstack al loro interno, ad esempio Rackspace e HP, mentre il cloud privato ed ibrido si ottiene attraverso il codice sorgente e le configurazioni sulle macchine host. Openstack contiene al suo interno dei Cloud Service, accessibili tramite interfacce standard RESTful, come le avviene per le altre soluzioni di infrastruttura analizzate nel capitolo 2.

L'architettura poggia concettualmente sulle unità fondamentali di Compute, Networking e Storage, ed essi sono specificati nei componenti che determinano la struttura di un sistema Openstack.
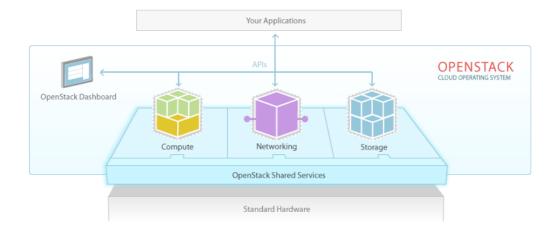
Fig. 46: Architettura logica di Openstack [53]

Openstack, giunto alla versione “Grizzly”, si fonda su un totale di sette componenti di base, illustrate in Fig. 47, comprensivi di quelli elencati precedentemente:
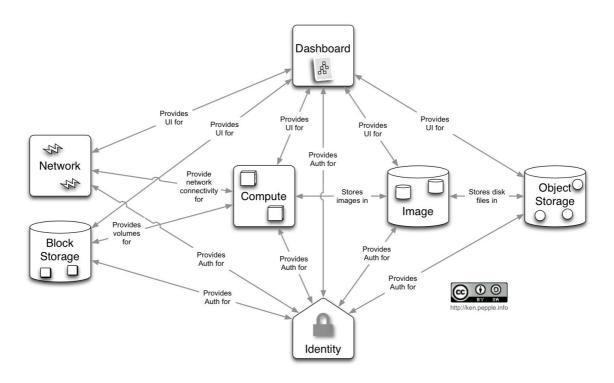
Fig. 47: Componenti del core di Openstack [55]

3.1.1 Nova (Compute)

Nova è un servizio di computazione offerto nel cloud. Come in EC2, le risorse di computazione sono accessibili sia attraverso interfacce web di amministrazione, sia attraverso delle API in modo programmatico. L'architettura è progettata per scalare in modo orizzontale sull'hardware standard, e per gestire automaticamente pool di risorse, attraverso diverse configurazioni di virtualizzazione come la bare-metal piuttosto che High-Performance Computing (HPC). Inoltre vi è il supporto per gli hypervisor più diffusi come Xen, ESX e KVM, e la scelta predefinita ricade su quest'ultimo. Una particolarità da sottolineare è il supporto di Openstack per l'architettura ARM (LPAE) [54], che insieme ad altre architetture supportate, rende meglio l'idea del concetto di piattaforma cloud computing ubiquo.

La gestione delle macchine virtuali merita un ulteriore accenno. Attraverso la Dashboard Horizon o interfacciandosi con i Web Service di Nova, è possibile gestire il ciclo di esecuzione delle istanze, le quali sono inoltre organizzate in Security Group per

il controllo degli accessi alle macchine virtuali. È presente inoltre la funzionalità di caching delle immagini nei nodi di computazione per un accelerare il processo di esecuzione delle istanze. Nova è composto da un insieme complesso di processi che insieme rendono possibili trasformare le richieste effettuate per mezzo dei servizi esposti, in operazioni relative alle istanze; i principali processi sono:

- **nova-api**: è il processo che riceve e gestisce le richieste effettuate attraverso le API. Supporta le Compute API di Openstack, ma anche quelle di Amazon EC2. Si fa inoltre carico dell'inizializzazione di alcuni attività di orchestrazione delle risorse;
- **nova-compute**: è il processo demone che si interfaccia con l'hypervisor per le operazioni di avvio e terminazione delle istanze, utilizzando le API esposte dagli hypervisor sottostanti (quelle di libvirt per KVM, XenAPI per Xen, VMwareAPI per ESX);
- **nova-schedule**: consulta la coda di richieste di esecuzione delle istanze e determina quale nodo di computazione debba eseguirla.

3.1.2 Swift (Object Storage)

È concettualmente simile ad Amazon S3, e pertanto permette di salvare file, anche se non è un file system. È piuttosto un servizio di storage ridondante. scalabile, e distribuito che permettere di salvare grosse moli di dati. Gli oggetti ed i file vengono salvati su più dischi all'interno del data center, ed Openstack funge da controller rivolto a garantire l'integrità dei dati dall'interno del cluster. Lo scaling è effettuato in modo orizzontale all'aggiunto di nuovi server ed Openstack si fa carico della reperibilità dei dati da altri nodi attivi, in caso di problemi su qualche disco o nodo. Swift comprende questi componenti:

- **Proxy server**: accetta le richieste di operazioni di storage quali caricamento di file, modifiche ai metadati e creazione dei container. Le richieste vengono effettuate tramite le Object API di Openstack oppure tramite semplici richieste HTTP;
- **Account servers:** gestiscono gli account definiti con Swift:
- **Container servers**: gestiscono il mapping dei container in Swift. Un container può essere una directory;
- **Object servers:** gestiscono gli oggetti sui nodi di storage. Un oggetto può essere un file.

3.1.3 Cinder (Block Storage)

Cinder è il servizio per la gestione dei volumi in Openstack, delle varie tipologie di volumi e dei volumi per le snapshot. Inizialmente, questa funzionalità era ricoperta da Nova, tuttavia successivamente è diventato un servizio a sé, come nelle maggiori soluzioni di IaaS, Il suo compito essenziale è quello di fornire volumi per le istanze e quindi l'interazione principale è con Nova. Espone delle API, mantiene un database interno per lo stato, e attraverso un suo scheduler seleziona il nodo migliore per lo storage.

3.1.4 Quantum/Neutron (Networking)

Neutron (già Quantum) è un sistema modulare, scalabile che espone delle API per le gestione della rete e degli indirizzi IP all'interno di Openstack. Fornisce connettività di rete as-a-Service fra le interfacce gestite da altri servizi in Openstack, come Nova. Permette agli utenti di creare le proprie reti e di associarvi delle interfacce di rete, attraverso la sua architettura a plugin e ad agenti che effettuano queste operazioni, a livello 3 dello stack ISO/OSI (IP). Viene garantita una larga interoperabilità grazie al supporto di sistemi di networking virtuale come quelli di Cisco, Nicira (ora VMware),

Open vSwitch. Anche Neutron ha un'interazione principalmente verso Nova e permette servizi avanzati di rete all'interno dell'infrastruttura di Openstack, ad esempio: LB-aaS, VPN-aaS, firewall-aaS, IDS-aaS, data-center-interconnect-aaS.

3.1.5 Keystone (Identity)

È un servizio che fornisce autenticazione ed autorizzazione per tutti i servizi in Openstack, come chiaramente illustrato in Fig. 38, che convergono su Keystone per policy, catalog, token ed autenticazione. Espone come tutti agli servizi delle API, ed ogni funzione in Keystone ha un backend modulare che permette diverse modalità di uso di un particolare servizio. Esempi sono il Key Value Stores (KVS), o il backend per LDAP;

3.1.6 Glance (Image Store)

Glance è un servizio per la gestione delle immagini. Attraverso esso, è possibile effettuare il discovery, la registrazione e la pubblicazione dei servizi per le stesse, nonché di quelli relativi ai dischi. Glance ricopre un ruolo centrale nel modello di infrastruttura di Openstack; gestisce le richieste relative alle immagini da parte di utenti finali (tramite Dashboard) o di Nova, per mezzo delle interfacce REST che espone tramite delle API, ed effettua il salvataggio dei file in Swift.

Utilizza un database interno per salvare i metadati relativi alle immagini e mantiene un registro per il recupero di queste informazioni. Fra i più, supporta file system, Amazon S3 e HTTP. Infine, rivestono un ruolo cruciale i suoi servizi di replicazione, che permettono di garantire disponibilità e consistenza all'interno del cluster di nodi per lo storage delle immagini. Supporta inoltre i più comuni tipi di immagini, quali ad esempio qcow2 per KVM, VMDK per VMware, VHD per Hyper-V o VDI per Virtualbox,

3.1.7 Heat (Orchestration)

Heat è il servizio utilizzato in Openstack per l'orchestrazione multipla e composita di applicazioni, conforme con OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) [56], e con il sistema di template di Amazon AWS CloudFormation (analizzato nel paragrafo 2.1.1.8), per mezzo delle API REST di Openstack e di API compatibili con Amazon CloudFormation Query API. [57] La Fig. 39 mostra lo schema di funzionamento dei componenti di Heat all'interno del cloud di Openstack:
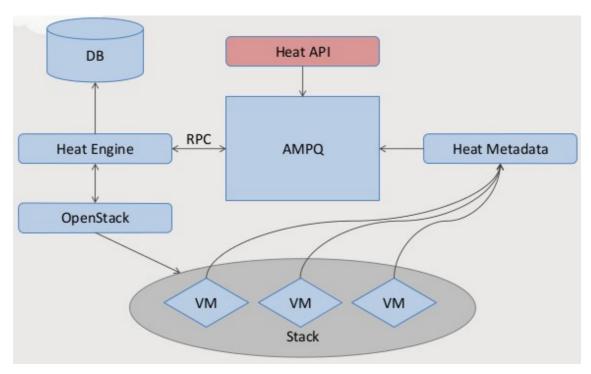
Fig. 48: Architettura e schema funzionamento di Heat [58]

- **heat** : è la CLI che comunica con il processo heat-api per richiamare le API di AWS CloudFormation, attraverso le API REST di Heat;

- **Heat API**: è un componente che fornisce delle API REST native in Openstack che smista tramite RPC le richieste fatte attraverso le suddette API verso il processo heat-engine ;
- **Heat Engine**: è il componente dove risiede la logica del servizio, il cui compito principale è quello di orchestrare le risorse dichiarate nei template e, tramite un meccanismo da eventi, fornire informazioni sulle richieste effettuate tramite le API;
- **Heat API CFN**: fornisce un sistema di Query API compatibile con AWS CloudFormation, anche queste inviate tramite RPC ad Heat Engine.

L'orchestrazione delle risorse in Heat avviene per stack. Attraverso il sistema di templating di CloudFormation si definiscono i parametri di stack, i mapping, le proprietà delle risorse e i valori di output utilizzando un file di testo scritto in JSON. Inoltre con il supporto delle API di AWS CloudWatch, analizzate sempre nel paragrafo 2.1.1.8, è possibile usufruire degli Alarm per interventi automatici sulle risorse monitorate, in base a delle regole definite nel template CloudFormation. CloudWatch è sostanzialmente un repository di metriche; attraverso le API compatibili e i template, il servizio Heat è in grado di ottenere delle statistiche basate su delle metriche definite ed è in grado orchestrare automaticamente le risorse, attraverso Autoscaling ad esempio. L'implementazione di CloudWatch in Heat contiene un sottoinsieme delle funzionalità di AWS CloudWatch, ed è principalmente richiesto per abilitare collezioni di metriche per High Availability (HA) delle applicazioni e AutoScaling delle risorse, oltre alla possibilità di definire Alarm personalizzati, come vedremo nel capitolo successivo.

I template CloudFormation-compliant in Heat possono integrare Puppet, un software di automazione che permette di definire degli stack di applicazione e di gestire tutto il ciclo di vita di un sistema operativo da eseguire come istanza nell'infrastruttura, dalla configurazione all'orchestrazione e al report. [60] È possibile, grazie al tool heat_jeos.sh presente in Heat, creare delle immagini Just Enough Operating System (JeOS) personalizzate e preparate per l'esecuzione in Openstack e l'orchestrazione

tramite Heat. Queste immagini vengono successivamente aggiunte a Glance ed orchestrate con Heat tramite il programma heat.

All'interno dei template CloudFormation è possibile inoltre definire delle dipendenze con altre istanze, in modo da definire un complesso di istanze in relazione fra di loro in base alle applicazioni che eseguono. Il deploy delle applicazioni con Heat avviene tramite Cloud Init, un software che gestisce l'inizializzazione delle istanze, e che comunica con il servizio Heat API CFN presente sulla macchina host (dove risiede Openstack), per le notifiche sul funzionamento dell'inizializzazione dell'istanza.

3.1.8 Horizon (Dashboard)

Horizon è una dashboard fornita come pannello di amministrazione accessibile via web, per una gestione semplice ed intuitiva di utenti e servizi di Openstack. È una Web Application modulare scritta in Python con il noto framework Django, accessibile via browser grazie a mod_wsgi del web server Apache. Permette di effettuare quasi tutte le operazioni altrimenti eseguibile dalla CLI, infatti è in rapido sviluppo per fornire un completo supporto dell'insieme delle operazioni in Openstack. Come visualizzabile in Fig. 49, la gestione delle istanze aggiunte tramite Glance è intuitiva, ed anche familiare per coloro i quali abbiano avuto esperienze con la Console di Amazon EC2.
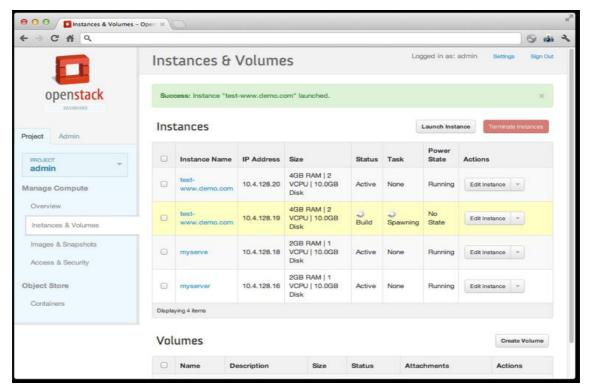
Fig. 49: Lancio di istanze tramite Horizon

3.1.9 Ceilometer (Metering)

Ceilometer è il nuovo sistema di metering di Openstack, che rimpiazzerà CloudWatch API. È uno strumento che misura la copertura del cloud, che mira a diventare un punto di raccolta per tutti i servizi erogati in Openstack, per effettuare delle misure da integrare in un sistema di pagamento/tariffazione delle risorse erogate.

3.1.10 Ironic (Bare-Metal)

Ironic è un driver di hypervisor per Nova per un approccio bare-metal al cloud di Openstack, la cui proprietà fondamentale è quella di non virtualizzare l'hardware. Le risorse fisiche sono esposte per mezzo delle API di Openstack, utilizzando un set di driver per fornire le funzionalità di Preboot Execution Imaging (PXE) ed Intelligent Platform Management Interface (IPMI). È alla base di TripleO, ovvero Openstack-over-Openstack, un progetto per fornire gestire le funzionalità classiche di Openstack, attraverso Openstack stesso bare-metal sulle risorse fisiche.

3.1.11 Juju (Orchestration)

Juju è un altro interessante servizio di orchestrazione, disponibile per la piattaforma cloud di Ubuntu Metal-as-a-Service (che offre servizi cloud su infrastruttura fisica, bare-metal), ma in grado di essere eseguito su più cloud quali lo stesso MaaS (su base Openstack Ironic), Openstack, Amazon EC2. È un progetto open source promosso da Canonical, e si basa esclusivamente su Ubuntu. Juju semplifica al massimo la descrizione delle risorse offerte nel cloud, con una sintassi molto intuitiva ed un sistema di template (Charm) che permette di definire le risorse e di definire delle relazioni fra di esse all'interno del cloud di riferimento, alla stregua di CloudFormation, ma con una sintassi di comandi più facile ed intuitiva. [61] La Fig. 50 illustra la copertura di Juju all'interno delle varie unità presenti in Openstack. Il concetto principale in Juju è la semplificazione nella descrizione delle risorse da orchestrare, quindi è molto facile definire dei servizi comuni innestati quali ad esempio Wordpress e MySQL e scalarli in qualsiasi momento con un solo comando, ma il vincolo sulla sola distribuzione Ubuntu come sistema di riferimento e una minore ricchezza sintattica rispetto a CloudFormation e CloudWatch, rendono Juju molto utile per deploy veloci, ma meno adatto alle strutture complesse realizzabili tramite Heat. Entrambi i servizi sono in continuo rapido sviluppo migliorando costantemente, e la compatibilità di entrambi con Amazon (Juju tramite istanza ubuntu su EC2, Heat tramite API CloudFormation e CloudWatch) permette l'interoperabilità fra più cloud, esplicitamente dichiarata in Juju.
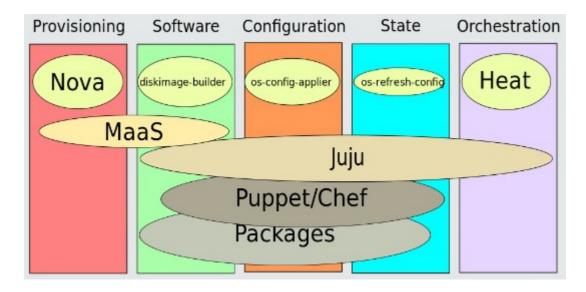
Fig. 50: Area di pertinenza di Juju nel cloud di Openstack [62]

3.2 OpenShift

OpenShift è la soluzione di cloud ibrido di piattaforma offerto da Red Hat, disponibile sia come PaaS pubblico sul sito ufficiale [63] sia come come PaaS ibrido nella versione Origin, dedicata alla community. [64] Supporta i runtime più comuni quali JavaEE6, Ruby, Python, Perl, MongoDB, MySQL e PostreSQL, tutti organizzati nel concetto di

Cartridge. Un Cartridge è essenzialmente un contenitore contenente un determinato runtime, predisposto per scalare e distribuirsi automaticamente sui nodi all'interno del PaaS di Openshift. La Fig. 51 illustra l'architettura su cui si fonda la platform:
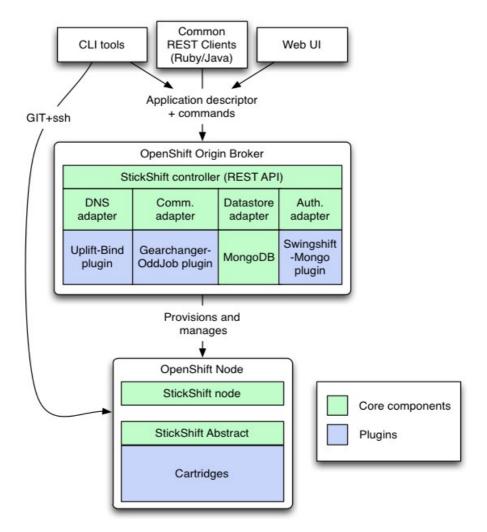
Fig. 51: Architettura e viste logiche in Openshift [64]

3.2.1 Broker

Openshift comprende due tipi logici di host: un *broker*, ed uno o più nodi. Concettualmente, le due componenti fondamentali della piattaforma sono: il Broker, che espone delle interfacce a servizi REST tramite delle API, e il Cartridge, citato precedentemente, che fornisce i runtime per gli ambienti di sviluppo delle applicazioni.

Il nodo Broker gestisce la creazione e la gestione delle applicazioni degli utenti

(orchestrazione delle applicazioni), ma anche l'autenticazione degli stessi attraverso un servizio di autenticazione, la comunicazione con i nodi appropriati, servizi di DNS per la platform. Tutte queste funzionalità sono fornite attraverso un rispettivo *adapter*, che come illustrato in Fig. 51, fa parte del core della struttura del nodo Broker. La persistenza interna è gestita tramite MongoDB, ed vi è anche la possibilità aggiungere dei plugin che estendono le funzionalità di base del Broker. È possibile interagire con esso attraverso la una Web Console fornita con la piattaforma, oppure per mezzo della CLI o delle API che il nodo espone.

3.2.2 Nodi e Gear

Nell'architettura di Openshift, un Nodo rappresenta una macchina dove vengono eseguite delle applicazioni. Questa macchina può essere fisica o virtuale, e contenere più Gear. Un Gear è un contenitore, formalmente un Container, nel quale sono eseguiti i Cartridge . Le risorse dei nodi sono suddivide i Gear attivi, distribuendo il carico di lavoro dell'applicazione come illustrato in Fig. 52
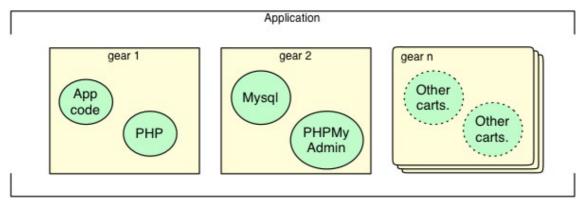
Fig. 52: Distribuzione delle applicazioni su più Gear [65]

Inoltre, come si evince chiaramente dalla figura, un Gear può eseguire più Cartridge per un'applicazione. In sostanza, un Gear è un'unità di CPU, di memoria primaria e

secondaria sul quale possono essere eseguiti dei componenti, e rappresenta pertanto un limite alle risorse da assegnare ad un Cartridge.

Il PaaS pubblico offerto da Red Hat prevede due tipi di Gear, Small e Medium, definiti nella Tabella 1:

Piano/Risorse	RAM	Disco	Costo
Small	512 MB	1 GB	Gratuito
Small	512 MB	6 GB	$0.04/ora
Medium	1 GB	6 GB	$0.10/ora

Tabella 1: Dimensioni e costo dei Gear [66]

Le istanze pubbliche che contengono questi Gear poggiano sull'infrastruttura EC2 di Amazon, pertanto la potenza di calcolo erogata fa riferimento alle ECU (discusse nel paragrafo 2.1.1.1), rispettivamente 1 e 2 ECU per i piani illustrati nella tabella.

È possibile tuttavia modificare queste impostazioni per i PaaS privati, scaricando il codice sorgente di Origin, nella classe Ruby

broker/config/environments/development.rb

```
config.openshift = {
   :domain_suffix => conf.get("CLOUD_DOMAIN", "example.com"),
   :default_max_gears => (conf.get("DEFAULT_MAX_GEARS", "100")).to_i,
   :default_gear_sizes => conf.get("DEFAULT_GEAR_SIZE", "small").split(","),
   :gear_sizes => conf.get("VALID_GEAR_SIZES", "small,medium").split(","),
       :default_gear_capabilities  =>  conf.get("DEFAULT_GEAR_CAPABILITIES",
"small").split(","),
   :community_quickstarts_url => conf.get('COMMUNITY_QUICKSTARTS_URL'),
  }
```

Listato 2: Definizione dei Gear nella classe development.rb

Oppure modificare nel Broker il file /etc/openshift/resource_limits.conf e definire delle configurazioni specifiche per i nuovi nodi, aumentando ad esempio il limite di RAM o

di disco, ma anche modificando le impostazioni relative alla potenza di calcolo.

3.2.2 StickShift API

La piattaforma fornisce un insieme di API per l'esposizione dei servi e per la gestione dei componenti interni, quelle appartenenti al core di Openshift sono le SticktStick API. Esse sono utilizzate sia nelle creazione della piattaforma, sia nella gestione dei Cartridge. Difatti, come visualizzabile in Fig. 51, sono presenti nella struttura logica del Broker e dei nodi, e mentre nel primo è presente un Controller dove risiede la logica di business della piattaforma, nei secondi sono presenti le API la gestione del ciclo di vita delle applicazioni e per i Gear associati ad esse.

3.2.3 Applicazioni Web

Le applicazioni vengono eseguite all'interno della piattaforma, ed accessibili dall'esterno generalmente tramite Internet, anche se non strettamente necessario. Riveste un ruolo fondamentale all'interno del PaaS di Openshift la gestione dei DNS, svolta del nodo Broker, che rappresenta l'unità di smistamento delle richieste esterne verso i nodi, seguendo la logica di distribuzione del carico di lavoro in base a Gear e Cartridge. Le organizzazioni che vogliono offrire il loro cloud di piattaforma, generalmente assegnano un dominio per la piattaforma, dal quale gestire richieste DNS su tramite domini di terzo livello, le applicazioni degli utenti. Pertanto, un'applicazione segue logicamente questa struttura:

- **Dominio**: corrisponde al namespace dell'utente, utilizzato per tutte le sue applicazioni. Il nome delle applicazioni dell'utente viene anteposto al dominio, per formare l'URL finale dell'applicazione;
- **Nome**: identifica il nome dell'applicazione;

- **Alias**: è possibile definire degli alias DNS nella piattaforma per creare degli indirizzi delle applicazioni più mnemonici e più corti;
- **Dipendenze**: l'utente definisce i Cartridge necessari all'esecuzione della proprio applicazione. Poiché si stratta di Web Application, viene definito un Cartridge di base per le applicazioni che è il Framework Cartridge, il cui compito principale è quello di servire pagine web. Inoltre, è possibile utilizzare un Cartridge opzionale detto Embedded Cartridge, per i Database.
- **Repository Git**: ciascuna applicazione mantiene un repository git all'interno della piattaforma, permettendo agli sviluppatori di modificare e pubblicare le loro applicazioni in maniera immediata e funzionale grazie noto al modello di condivisione e sviluppo del codice offerto da git.

La creazione delle applicazioni può avvenire in due modi: attraverso la procedura standard del processo di sviluppo mediato dal Broker e gestito dall'utente, come illustrato in Fig.53, oppure attraverso Jenkins CI [67], uno strumento open source di integrazione continua integrato nella piattaforma.
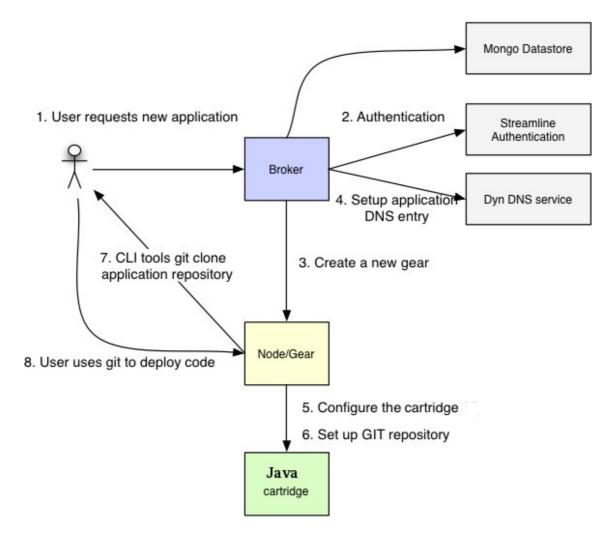
Fig. 53: Processo standard di creazione delle applicazioni [65]

L'utente effettua la richiesta di creazione di un'applicazione attraverso la console web di Openshift o attraverso gli OpenShift Commandline Tools (RHC) , raggruppati appunto nel programma rhc . Tale richiesta arriva al Broker, il quale autentica l'applicazione ed imposta un entry DNS per la stessa, come discusso precedentemente. Successivamente, il Broker crea i Gear, ed il controller del nodo contenente gli stessi configura i Cartridge richiesti per l'applicazione (nell'esempio quello di Java), ed imposta il repository Git. Infine, l'utente viene informato sull'avvenuta creazione del repository e può effettuare il deploy dell'applicazione tramite Git stesso.

La creazione delle applicazioni all'interno della piattaforma avviene in maniera

dichiarativa per mezzo di un file YAML, il quale viene utilizzato per descrivere l'architettura desiderata per l'applicazione, nonché l'applicazione stessa tramite i campi analizzati precedentemente. Il Broker utilizza tale file per creare o modificare l'applicazione e tramite le Stickshift API è possibile manipolare i campi descritti del file YAML. Un esempio di file descrittore è il seguente:

```
Name: Esempio

Version: 1.0

Requires: jbossas-7, mysql

Connections:

  - jbossas-7, mysql

Group override:

  - jbossas-7, mysql
```

Listato 2: Esempio descrizione di Web Application su base JBoss

Openshift Origin prevede di default che il codice e i binari risultati risultino entrambi nel processo di deploy della piattaforma, facilitando lo sviluppo condiviso e la revisione del codice; è anche possibile un'integrazione con Github. Tuttavia, qualora non si volesse depositare il proprio codice sulla piattaforma, è possibile copiare manualmente il binario nella directory dove risiedono i binari .

3.2.4 Auto scaling

In Openshift è previsto lo scaling orizzontale delle applicazioni, grazie ad un HighAvailability Proxy (HAProxy) presente sul primo Gear associato, che funge da Load Balancer e da end-point per i deploy effettuati tramite Git (push). Le richieste HTTP vengono inoltrate ad HAProxy, il quale provvede a smistarle sul Gear che possiede lo strato web dell'applicazione. HAProxy è nella fattispecie un Cartridge:

quando un utente della piattaforma effettua il push della sua applicazione su Gear di HAProxy, automaticamente viene eseguito un push su tutti i Gear contenenti lo strato web dell'applicazione. Tecnicamente, ciò che scala, a fronte del carico di richieste ricevute, è il Gear. Uno scenario tipico è illustrato in Fig. 54.

Il processo di scaling delle risorse all'interno della piattaforma comprende:

- Viene creata una nuova entry DNS per ogni nuovo Gear creato, il cui URL sarà:
    - <ID_CASUALE>-<NAMESPACE_UTENTE>.<DOMINIO_CLOUD>
- La logica all'interno del controller copia le impostazioni del Gear originale verso quello nuovo;
- Il Broker comunica con HAProxy le configurazioni per il Gear che sta effettuando lo scaling.
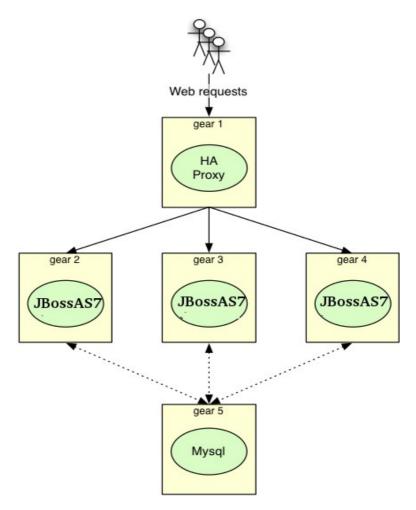
Fig. 54: Esempio di auto scaling orizzontale in Openshift [65]

3.2.5 Cartridge

Come già accennato, un Cartridge è un contenitore dei runtime utilizzati nei per fare girare le applicazioni nei rispettivi Gear. Tutta l'architettura di Openshift è modulare in sé, ed è possibile estendere le funzionalità della piattaforma grazie alla creazione di nuovi Cartridge. Attualmente, le specifiche per la scrittura di nuovi Cartridge sono alla versione 2, reperibili sul sito della community. [68]

Tra le funzionalità più interessanti presenti nella definizione dei Cartridge vi è la

possibilità di effettuare snapshot delle applicazioni e il processo di scaling delle stesse.

- **Snapshot**: la piattaforma supporta un meccanismo snapshot/restore per le applicazioni, che avviene tramite un processo di stop/start dei Gear dove risiede. È possibile personalizzare il processo di snapshot in termini di selezione o esclusione di alcuni file da inserire in un archivio tar.gz, all'interno delle definizione del Cartridge. Difatti dopo lo stop del Gear, avviene un processo di pre-snapshot effettuato per ogni Cartridge nel Gear, per controlli quali la serializzazione dello snapshot. Allo stesso modo, prima del restart del Gear, avviene un processo di post-snapshot che effettua i controlli sullo stato della snapshot inerente a ciascun Cartridge nel Gear;
- **Scaling**: come già accennato, ciò che scala effettivamente nella piattaforma è il Gear, il quale contiene dei Cartridge, e l'insieme dei Gear associati ad un'applicazione permette di effettuare lo scaling della stessa, in base al carico di richieste e al numero di risorse associate al Gear, definiti all'interno della piattaforma come discusso nel paragrafo 3.2.2. Il codice delle applicazioni di cui è stato effettuato il push, viene sottoposto ad un processo di build che comprende, secondo il ciclo di vita standard, il build stesso delle applicazioni ed il deploy. Ciò che differenzia un'applicazione scalabile dalle altre, è la fase di deploy nella quale grazie ad HAProxy vengono coinvolti dei Gear secondari per l'avvio sugli stessi, in ordine di esecuzione, dei Cartridge secondari e di quello primario dell'applicazione, ed infine il deploy di quest'ultimo sul Gear primario.

3.3 High Availability

Esistono due tipi di sistemi di High Availability: di istanza e di servizio. Le considerazioni precedenti, fatte per HAProxy, fanno riferimento ad un HA di servizio. Più in generale, un High Availability (HA) è un sistema che prevede il corretto funzionamento di un server, in base a delle misurazioni sui tempi di risposta, anche di fronte a grossi carichi di richieste. Per definire meglio il concetto, è possibile esprimerlo attraverso l'Eq. 3.1:

$$A = \frac{MTBF}{MTBF + MTTR}$$

(3.1)

dove A corrisponde al grado di disponibilità del servizio, espresso in termini percentuali, MTBF è il tempo medio fra guasti che si verificano e MTTR è il tempo massimo necessario a riparare un particolare guasto. Se MTTR tende a zero, la disponibilità è quasi totale, mentre al crescere di MTBF, diminuisce l'impatto di MTTR su A.

Le misure sulla disponibilità di un sistema coinvolgono una serie di decisioni sul tipo di metrica, sulla validità temporale della stessa e sulla possibilità che essa si adatti ad un processo di miglioramento del sistema. In questo senso il concetto di High Availability, viene espresso come tolleranza ai guasti, considerando dei fattori di rischio quale quello di downtime, ovvero di non disponibilità. Ed i rischi hanno dei costi. Pertanto viene espressa un ulteriore formulazione per ridurre il rischio di downtime:

$$S = R_b - R_a \quad (3.2)$$

$$ROI = S / C_M \quad (3.3)$$

dove S è il risparmio ottenuto, R_b il rischio prima di applicare delle misure protettive, R_a il rischio dopo averle applicate, ottenendo il Return Of Investement (ROI) in termini percentuali come il risparmio ottenuto diviso C_M, ovvero il costo di implementazione. Calcolare il rischio non è semplice, bisogna tener conto di fattori come la probabilità che si verifichi il guasto, la durata media dello stesso e l'impatto sui

sistemi coinvolti. Nella soluzione proposta nel capitolo successivo, vengono coinvolti due sistemi di HA, uno di istanza orchestrato da Heat, e l'altro di servizio orchestrato dal nodo Broker. Un doppio sistema di HA permette di garantire una maggiore tempestività nella risoluzione dei problemi e nella riduzione del rischio di guasto, grazie all'esecuzione di istanze che prevengono un carico eccessivo dei sistemi.

4. RDO e Openshift Origin Per Sviluppo Web Enterprise

A-wop-bom-a-loo-mop-a-lomp-bom-bom! [69]

Dopo aver analizzato tutte le soluzioni possibili nel cloud, la scelta dell'infrastruttura e della piattaforma più adatta ad un ambiente di sviluppo Web Enterprise è ricaduta sui due progetti analizzati nel capitolo precedente, rispettivamente Openstack ed Openshift. Entrambe le soluzioni sono promosse e supportate da Red Hat, la quale è anche autrice della seconda. La versione di Openstack utilizzata è la Grizzly, analizzata in dettaglio nel precedente capitolo, pacchettizzata su RPM e presentata come Red Hat Openstack (RDO) su base RHEL e derivate (CentOS, Fedora). La versione di Openshift adottata invece è la versione open source denominata Openshift Origin, pacchettizzata anch'essa su RPM per RHEL e derivate. Lo scopo finale è stato quello di raggiungere l'orchestrazione delle risorse con Heat e l'autoscaling delle applicazioni tramite HAProxy, mediante un piccola configurazione dimostrativa, realizzando così il paradigma PaaS over IaaS. Prima di cominciare a descrivere la configurazione, è necessaria fare una premessa su cosa si intende per sviluppo Web Enterprise e quali soluzioni sono state considerate sulla base della scelta infrastrutturale proposta.

4.1 Sviluppo Web Enterprise

Vi sono molte argomentazioni sul significato di applicazione web enterprise, termine familiare in IT. Tipicamente si riferisce ad un processo di sviluppo di applicazioni Web che segue determinati processi aziendali, ma la definizione più recente e concisa si trova in [68] e può essere interpretata così: si definiscono Web enterprise tutte le applicazioni coerenti con le logiche di business online delle organizzazioni che le realizzano.

La definizione è astratta e indipendente dal tipo di tecnologia usata, tuttavia solitamente si fa riferimento ad ambienti di sviluppo Java Enterprise Edition (JavaEE), poiché nel tempo, JavaEE ha consolidato in questo settore diverse soluzioni adottate da un ampio numero di organizzazioni. Attualmente vi sono diverse soluzioni diverse da Java, come Ruby, Python, .NET, Javascript, che offrono le stesse potenzialità, ciononostante in questa discussione analizzeremo Java e il suo consolidato platform server JBoss, integrato in Openshift come Cartridge, il quale fornisce funzioni peculiari nel contesto enterprise quali autenticazione, logica di business, livelli di persistenza, load balancing e scalabilità.

4.1.1 JBossAS7

JBoss Enteprise Application Server 7 è un application server open source per Java, conforme a tutte le specifiche di JavaEE6. [70] JBoss viene fornito come piattaforma di sviluppo enterprise, nella quale effettuare il deploy delle proprio applicazioni web, secondo delle logiche di business. La Fig, 55 illustra l'architettura della piattaforma JBoss, attraverso le sue unità fondamentali.
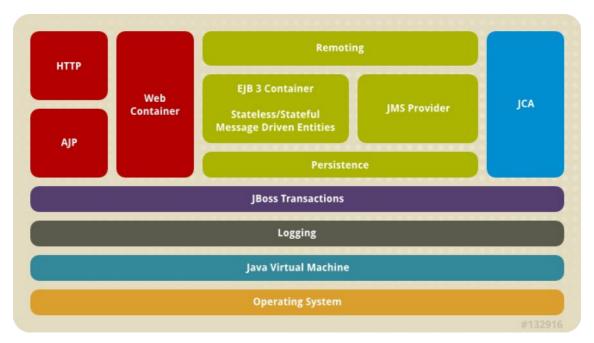
Fig. 55: Architettura di JBoss EAP

La funzionalità chiave in JBoss AS7 è la possibilità di gestire più server da un singolo punto di controllo. Nella versione corrente, viene introdotto il concetto di dominio, inteso come collezione di server. Tali domini possono essere diverse macchine fisiche o virtuali con istanze di AS7 su un certo host, sotto il controllo di un Host Controller. Gli Host Controller interagiscono con i Domain Controller per controllare il ciclo di vita delle istanze di JBoss e nella gestione di esse insieme al Domain Controller stesso.

Un'altra funzionalità utile da menzionare, utile ai fini di questo elaborato, è la predisposizione di JBoss ad HighAvailability, funzionalità utilizzata nello scaling orizzontale proprio da Openshift. Le configurazioni che rendono possibile la clusterizzazione di JBoss sono molteplici. Tra le più importanti: mod_cluster per il front end (Apache), le configurazioni per i Container EJB, per il Load Balancing e JNDI. [71]

4.2 Data Center

La struttura fisica sulla quale si è realizzata l'infrastruttura si basa su un rack composto da 4 unità di calcolo, presente nel laboratorio di Grid Computing. Il sistema è stato installato su una singola macchina HP ProLiant DL360 Generation 5 (G5) con queste specifiche hardware:

Fig. 56: HP ProLiant DL360 Generation 5 (G5)

- Processore Intel(R) Xeon(R) CPU 5160 3.00GHz
- Memoria RAM 2GB
- Hard Drive iSCSI 76 GB

Tale configurazione hardware non è la più indicata per un'infrastruttura di produzione e di testing a causa delle limitazioni della memoria RAM e del disco, ma risulta sufficiente per l'illustrazione dimostrative delle funzionalità contenuti in tutti i componenti installati. Fornito di due uscite ethernet, una interna per lo switch del rack e l'altra per accesso ad Internet, è reperibile su http://moon4.deis.unical.it

4.3 RDO

Il sistema base scelto sul quale poggia l'infrastruttura è CentOS 6.4 per architetture a 64 bit [72], installato nella versione NetInstall a causa di un bug nell'installer Anaconda della versione standard che comprometteva l'installazione a metà del processo. [73]

CentOS è una distribuzione Linux di classe Enterprise, derivata dai sorgenti di RHEL, e rappresenta la versione community della nota distribuzione Linux adottata nelle soluzioni enterprise. La differenza fra le due versioni community ed enterprise risiede quasi esclusivamente nell'assenza di assistenza tecnica e di garanzia nell'esecuzione del sistema da parte della prima, e questo fa di CentOS una soluzione altamente affidabile e consigliata in ambienti server.

Come già accennato, la versione di Openstack utilizzata è la Grizzly, pacchettizzata su RPM e disponibile per la famiglia di distribuzioni Red Hat come RDO, su un repository appositamente creato per una gestione semplificata del processo di deploy di Openstack.

4.3.1 Installazione di Openstack

RDO prevede, come prerequisiti hardware, un minimo di 2GB di memoria disponibile, che combacia con l'hardware a disposizione, e il processo di installazione di Openstack diviene semplificato rispetto alla normale procedura di installazione da sorgenti [74] grazie all'installazione di un RPM, il qualche richiama anche tutte le altre dipendenze ed una serie di altri script necessari. È bene ricordare che un pacchetto RPM corrisponde ad un binario di installazione di un programma, e nei sistemi di pacchettizzazione quali RPM è prevista una gestione dei programmi a pacchetti e dipendenze. A sua volta un pacchetto RPM, oltre ad installare altri pacchetti necessari per le sue funzioni, può richiamare degli script di post-install utilizzati per effettuare delle configurazioni dinamiche aggiuntive, altrimenti non possibili in un contesto statico.

I pacchetti RPM possono essere installati manualmente, attraverso il programma rpm, oppure attraverso un sistema di gestione coordinata di pacchetti basato su repository, chiamato yum.

La procedura di installazione di Openstack si riassume semplicemente nell'installazione di un meta-pacchetto che contiene le informazioni per il download di Openstack con tutte le sue dipendenze, e l'utilizzo di Packstack per l'esecuzione dei task di configurazione di Openstack, altrimenti da fare manualmente. Packstack è un programma di utilità che usa dei moduli Puppet per configurare automaticamente tramite SSH le installazioni di Openstack. Attraverso l'esecuzione di tre comandi, grazie al sistema appena descritto, è possibile avere un Openstack Grizzly basato su RPM funzionante e pronto all'uso:

```
yum install -y http://rdo.fedorapeople.org/openstack/openstack-grizzly/rdo-release-grizzly.rpm

yum install -y openstack-packstack

packstack --allinone
```

Listato 3: Installazione di Openstack Grizzly (RDO)

Openstack si predispone ad installazioni multi-host (nodi di computazione), ed è possibile utilizzare Puppet per gestire le configurazioni remote di più host. Questo procedura prevede l'installazione di un solo nodo fisico computazionale, ma è possibile modificare gli script di Puppet, oppure aggiungere altri nodi successivamente all'installazione per ottenere un'installazione di Openstack per la gestione di più unità fisiche. Al termine dell'installazione, che richiede alcuni minuti, vi è la possibilità di accedere ad Horizon, l'interfaccia web di gestione di Openstack analizzata nel paragrafo 3.1.8, per creare le utente necessarie al deploy delle istanze e per accedere alla maggior parte delle funzionalità di IaaS offerte.
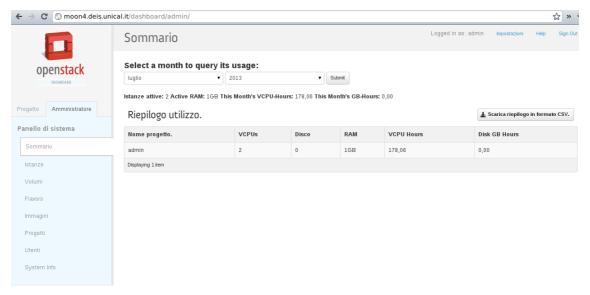
Fig. 57: Screenhot della pagina iniziale in Horizon

Horizon fornisce delle statistiche immediate sulle quote di utilizzazione delle istanze, nonché una semplice gestione delle stesse, come avviene con la Console di Amazon EC2. Purtroppo non è possibile usufruire di tutte le funzioni di Openstack dalla dashboard, poiché non ancora integrate, nonostante il continuo work-in-progress.

L'installazione di Openstack effettuata tramite Packstack e Puppet, prepara l'ambiente per la gestione dei componenti. Poiché, come illustrato in Fig. 47, Keystone funge da servizio per l'autenticazione dei componenti di Openstack, viene creato un utente admin di default con una password casuale, salvata nel file keystonerc_admin, presente nella directory dell'utente di sistema che ha effettuato l'installazione (con privilegi di amministratore). Tale password è ovviamente modificabile sia dalla CLI attraverso il tool keystone, sia da Horizon graficamente in maniera semplice.

4.3.2 Configurazione di Heat su RDO

Come già discusso, Heat è il servizio di orchestrazione ufficiale di Openstack. È abbastanza recente come progetto e non essendo inizialmente integrato nel core di Openstack, vi sono difficoltà nella configurazione dello stesso su RDO. Heat è in costante evoluzione e sviluppo, ed è difficile mantenere una coerenza funzionale fra la versione pacchettizzata di Openstack e il repository Github sul branch master del servizio di orchestrazione. Esiste un branch apposito per soluzioni stabili come RDO, dal nome grizzly/stable, ma il processo di bugfix e l'inserimento di nuove feature è sottoposto ad un lento processo di revisione, che avviene se sollecitata dal team di sviluppo, al quale, ad esempio, ho segnalato un bug.

Poiché l'installazione effettuata con Packstack non installa e configura anche Heat, è necessario effettuare la procedura manualmente. La guida ufficiale fa riferimento all'installazione su Fedora [75] che però non funziona su RDO, per il quale è necessaria una configurazione manuale particolare apparsa recentemente sul Wiki di RDO [76] alla quale ho apportato anche il mio contributo. La configurazione di Heat prevede l'aggiunta dell'utente heat tramite Keystone, sul quale come abbiamo visto, convergono tutti i servizi di Openstack, ed il mantenimento della persistenza interna del servizio su MySQL. È possibile recuperare tutte le impostazioni e le password casuali utilizzate da Packstack durante l'installazione automatizzata nel file di answer generato nella directory dell'utente che ha lanciato l'installazione degli RPM.

```
heat-db-setup rpm -y -r ${MYSQL_ROOT_PASSWORD_IN_ANSWER_FILE} -p $
{PASSWORD_DB_HEAT_CUSTOM}
source keystonerc_admin
keystone user-create --name heat --pass ${PASSWORD_USER_HEAT_CUSTOM}
keystone user-role-add --user heat --role admin --tenant services
keystone service-create --name heat --type orchestration
keystone service-create --name heat-cfn --type cloudformation
keystone endpoint-create --region RegionOne --service-id ${HEAT_CFN_SERVICE_ID}
--publicurl "http://${HEAT_CFN_HOSTNAME}:8000/v1" --adminurl "http://$
```

```
{HEAT_CFN_HOSTNAME}:8000/v1" --internalurl "http://$
{HEAT_CFN_HOSTNAME}:8000/v1"
keystone endpoint-create --region RegionOne --service-id ${HEAT_SERVICE_ID}
--publicurl "http://${HEAT_HOSTNAME}:8004/v1/%(tenant_id)s" --adminurl "http://$
{HEAT_HOSTNAME}:8004/v1/%(tenant_id)s --internalurl "http://$
{HEAT_HOSTNAME}:8004/v1/%(tenant_id)s"
```

Listato 5: Configurazione manuale di Heat

Come illustrato in Fig. 58, attraverso queste operazioni si ottiene l'aggiunto dell'utente heat al Keystone, abilitato dunque nell'infrastruttura dal sistema di autenticazione, insieme a tutti gli altri servizi elencati.

Utenti

Filter | Filter | Create User

	User Name	Email	ID utente	Abilitato	Azioni
	glance	glance@localhost	03a37354fd9d43ff869216a83b77485f	True	Modifica More
	swift	swift@localhost	7e65bb177e054aa5bf05c83316b47276	True	Modifica More
	cinder	cinder@localhost	1abb20acd29e46e28ab0aefca836d852	True	Modifica More
	nova	nova@localhost	438964e3f58c48e493fc2354a9a57c87	True	Modifica More
	admin	test@test.com	5afbcb6edeac4697af0a86789577eb1f	True	Modifica More
	heat	None	65b513e55f2741269c22c9e846cb575f	True	Modifica More

Fig. 58: Utente heat aggiunto al Keystone

Nel capito precedente è stato accennato il sistema di template e di Alarm utilizzato da Openstack e da Heat per la definizione delle risorse da orchestrare e degli eventi associate ad esse. Al fine di abilitare questo sistema, è necessario configurare i Web Service compatibili di CloudFormation e CloudWatch inserendo le impostazioni appena create per il Keystone e configurando gli indirizzi di rete, attraverso i quali possono essere usufruiti tali servizi. Bisogna pertanto aggiornare i rispettivi file di configurazione delle API di Heat, di CloudFormation e CloudWatch presenti in /etc/heat

ovvero heat-api-paste.ini heat-api-cfn-paste.ini e heat-api-cloudwatch.ini come segue:

```
admin_tenant_name = services
admin_user = heat
admin_password = ${PASSWORD_USER_HEAT_CUSTOM}
```

Listato 6: Configurazione Heat API

In questa configurazione è stato scelto KVM come hypervisor, utilizzato da Nova per mezzo di libvirt per erogare il servizio di computazione on-demand, tramite virtualizzazione. KVM è la soluzione predefinita in RDO ed Openstack, sebbene, come discusso precedentemente, non vi siano limitazioni sull'hypervisor da utilizzare, grazie all'astrazione fornita da libvirt e alla natura open source di Openstack stesso. Per ottenere gli indirizzi IP virtuali che permettano di mappare le istanze per la comunicazione fra i servizi sopracitati, bisogna consultare il file di configurazione di Nova, reperibile in /etc/nova/nova.conf. Tale file, contiene sia le impostazioni di computazione (appunto il tipo di hypervisor), sia la classe di indirizzi IP virtuali sulla quale vengono eseguite le istanze.

```
fixed_range=192.168.32.0/22
connection_type=libvirt
```

Listato 7: Maschera di indirizzamento delle istanze definita in nova.conf

Il networking è gestito da nova-network, denominato Libvirt Flat Networking, poiché Packstack su RDO non utilizza al momento Neutron. Vengono utilizzate delle interfacce virtuali mappate nel sistema, e le istanze di Nova comunicano fra di loro tramite l'interfaccia br100 creata sulla maschera di rete definita in nova.conf. In questa configurazione di default, sono disponibili 1022 host per le istanze virtuali, le quali hanno accesso alla rete esterna grazie al bridge. Per abilitarne la comunicazione in Heat, si deve modificare l'impostazione di default presente in /etc/heat/heat-engine.conf ed inserire la classe di IP utilizzata da Nova. Bisogna inoltre verificare che la password del DB MySQL utilizzato da Heat, sia conforme a quella impostata nella configurazione iniziale del servizio, illustrata nel Listato 2.

```
# URL for instances to connect for metadata
# ie the IP of the bridge device connecting the
# instances with the host and the bind_port of
# the CFN API
# NOTE : change this from 127.0.0.1 !!
heat_metadata_server_url = http://192.168.32.1:8000
# URL for instances to connect for notification
# of waitcondition events (ie via cfn-signal)
# e.g the IP of the bridge device connecting the
# instances with the host and the bind_port of
# the CFN API
# NOTE : change this from 127.0.0.1 !!
heat_waitcondition_server_url = http://192.168.32.1:8000/v1/waitcondition
# URL for instances to connect for publishing metric
# data (ie via cfn-push-stats)
# e.g the IP of the bridge device connecting the
# instances with the host and the bind_port of
# the heat-api-cloudwatch API
# NOTE : change this from 127.0.0.1 !!
heat_watch_server_url = http://192.168.32.1:8003
# The namespace for the custom backend. Must provide class Clients which will
be
# imported. Defaults to OpenStack if none provided.
# cloud_backend=deltacloud_heat.client
sql_connection = mysql://heat:${PASSWORD_DB_HEAT_CUSTOM} @localhost/heat
```

Listato 8: Configurazione di Heat Engine

La configurazione del servizio termina con la creazione di ruolo apposito da associare a delle utenze create da Heat per la ricezione dei dati ottenuta tramite meccanismi di *wait condition* e *signaling* utilizzati per l'orchestrazione delle risorse definite nei template, e con il riavvio del servizio Heat.

```
keystone role-create --name heat_stack_user
cd /etc/init.d && for s in $(ls openstack-heat-*); do chkconfig $s on && service $s start;
done
```

Listato 9: Procedure post-configurazione Heat

Come illustrato nel Listato 8, i Web Service utilizzano le porte 8000 e 8003 per il processo di comunicazione RESTful nell'orchestrazione delle risorse mediante le API compatibili di CloudFormation. È bene verificare che tali porte non siano bloccate dal alcun firewall sulla macchina host che esegue Openstack, come ad esempio avviene su RDO per la presenza di SELinux ed iptables, il quale viene configurato automaticamente, ma non presenta (almeno su Centos 6.4 ed RDO Grizzly) una chain per abilitare la comunicazione su quelle porte da parte delle istanze. A tale fine, si può aggiungere in /etc/sysconfig/iptables due regole per l'abilitazione della comunicazione fra le istanze su quelle determinate porte:

```
-A INPUT -i br100 -p tcp --dport 8000 -j ACCEPT
-A INPUT -i br100 -p tcp --dport 8003 -j ACCEPT
```

Listato 10: Abilitazione porte CFN API su iptables

4.3.3 Orchestrazione di Openshift

Una volta configurato Heat, si può verificarne il corretto funzionamento tramite un template qualsiasi disponibile nel progetto heat-teamples [77], a cui ho partecipato durante la stesura di questo elaborato. È possibile definire dei template con dipendenze fra le istanze e, tramite CloudWatch, ottenere delle notifiche o meglio Alarm durante l'esecuzione delle stesse. Le risorse da orchestrare si basano su delle immagini, che vengono create attraverso più strumenti, per poi essere aggiunte a Glance. Al fine di poter effettuare un deploy dimostrativo di una PaaS su base Openshift, e su un IaaS su base Openstack con orchestrazione via Heat, è possibile utilizzare dei template appositi per Openshift Origin presenti in heat-templates, i quali prevedono la presenza di una coppia Broker/Nodo, con la possibilità di effettuare auto scaling tramite le API di Amazon CloudFormation su base computazionale Openstack Nova. Per effettuare

questo deploy, è necessario creare delle immagini di sistemi derivati da Red Hat, configurati automaticamente. Nel caso specifico, esistono delle diverse soluzioni di base Fedora, e nella configurazione dimostrativa, è stata provata una soluzione basata su Fedora 18 e 19.

4.3.3.1 Creazione immagini

È possibile creare le immagini alla base del processo di orchestrazione delle risorse in diversi modi, attualmente si usano questi metodi:

4.3.3.1.1 heat_jeos.sh : Fedora 18

heat_jeos.sh è uno script contenuto in heat-templates che permette la creazione di immagini JeOS in base a dei template TDL. Utilizza Oz [78], un sistema di creazione di immagini personalizzate che definisce appunto i template TDL, per il quale è richiesta l'installazione clonando la versione più aggiornata, poiché la versione RPM su RDO non funziona con gli script per Openshift presenti su heat-templates.

La creazione dell'immagine avviene mediante l'invocazione dello script, il quale crea tramite qemu-img un immagine QCOW2 pronta per la virtualizzazione con QEMU/KVM tramite Nova.

```
./heat_jeos.sh ../jeos/F18-x86_64-openshift-origin-broker-cfntools.tdl F18-x86_64
openshift-origin-broker-cfntools
./heat_jeos.sh ../jeos/F18-x86_64-openshift-origin-node-cfntools.tdl F18-x86_64
openshift-origin-node-cfntools
```

Listato 11: Creazione immagini Openshift tramite heat_jeos.sh

4.3.3.1.2 diskimage-builder : Fedora 18

Il metodo “ufficiale” per la creazione delle immagini della coppia Broker/Nodo di Openshift su heat-templates prevede l'interazione di Oz, con diskimage-builder, un tool utilizzato in TripleO. Per eseguire la procedura completa, bisogna lanciare questi comandi:

```
git clone https://github.com/stackforge/diskimage-builder.git
git clone https://github.com/stackforge/tripleo-image-elements.git
mkdir $HOME/tmp
export ELEMENTS_PATH=tripleo-image-elements/elements:heat-templates/openshift-origin/elements
TMP_DIR=$HOME/tmp DIB_IMAGE_SIZE=5 diskimage-builder/bin/disk-image-create --no-tmpfs -a amd64 vm fedora openshift-origin-broker -o F18-x86_64-openshift-origin-broker-cfntools
TMP_DIR=$HOME/tmp DIB_IMAGE_SIZE=20 diskimage-builder/bin/disk-image-create
--no-tmpfs -a amd64 vm fedora openshift-origin-node -o F18-x86_64-openshift-
origin-node-cfntools
```

Listato 12: Creazione immagini Openshift tramite diskimage-builder

```
<template>
  <name>F18-x86_64-openshift-origin-broker-cfntools</name>
  <os>
    <name>Fedora</name>
    <version>18</version>
    <arch>x86_64</arch>
    <install type='iso'>
      <iso>file:/var/lib/libvirt/images/Fedora-18-x86_64-netinst.iso</iso>
    </install>
  </os>
  <description>OpenShift Origin Broker</description>
  <files>
    <file name='/etc/yum.repos.d/puppetlabs-products.repo'>
[puppetlabs-products]
..
  <commands>
    <command name='lockroot'>
passwd -l root
    </command>
```

```
    <command name='user'>
/usr/sbin/useradd ec2-user
echo -e 'ec2-user\tALL=(ALL)\tNOPASSWD: ALL' >> /etc/sudoers
    </command>
    <command name='packages'>
yum -y update fedora-release;
yum -y install yum-plugin-fastestmirror cloud-init python-psutil python-pip;
..
    </command>
  </commands>
```

Listato 13: Template TDL per Openshift Broker

Nel Listato 13 sono state evidenziate in grassetto delle informazioni utili a comprendere il processo di creazione e configurazione delle istanze tramite le API di Heat.

1. Viene mostrata la directory di default dove risiedono le immagini per QEMU/KVM, la quale viene usata come riferimento dagli altri servizi in Openstack per il caricamento delle immagini;
2. Viene dichiarato l'utente di base ec2-user il quale rappresenta un utenza di istanza, che accende alla macchina virtuale tramite autenticazione a chiave pubblica con le credenziali aggiunte al Keystone precedentemente (e dichiarate successivamente nel deploy con Heat);
3. Tra gli altri, viene installato sull'istanza, come processo di post-configurazione, cloud-init, uno strumento che gestisce le inizializzazioni delle istanze compatibili con EC2. [79]

4.3.3.1.3 Vagrant

Anche attraverso Vagrant [80] si possono ottenere delle immagini preconfigurate per una coppia di istanze Broker/Nodo. Vagrant è un altro sistema per la creazione di ambienti virtuali tramite templating. La guida ufficiale di Openshift suggerisce questo metodo per avere una coppia di istanze su base Fedora 19, da provare in un ambiente di virtualizzazione, sebbene l'immagine fornita sia legata a VirtualBox.

4.3.3.2 Deploy con Heat

Una volta generate le immagini QCOW2 nella directory di libvirt, attraverso una delle tecniche appena descritta, bisogna aggiungerle a Glance come segue:

```
glance add name=F18-x86_64-openshift-origin-broker-cfntools is_public=true
disk_format=qcow2 container_format=bare < /var/lib/libvirt/images/F18-x86_64-openshift-
origin-broker-cfntools.qcow2

glance add name=F18-x86_64-openshift-origin-node-cfntools is_public=true
disk_format=qcow2 container_format=bare < /var/lib/libvirt/images/F18-x86_64-openshift-
origin-node-cfntools.qcow2
```

Listato 14: Aggiusta delle immagini a Glance

Se la procedura viene eseguita correttamente, Glance dovrebbe avere mappato le immagini come segue:

Fig. 59: Immagini Openshift aggiunte a Glance

Dopo avere mappato le immagini all'interno dell'infrastruttura, è ora possibile orchestrare la coppia di Broker/Nodo con Heat, attraverso un template CloudFormation o YAML compatibile, e gestirne gli eventi con le API compatibili con CloudWatch. Al fine di abilitare l'accesso SSH alle istanze, è necessario inserire all'interno di Nova, la chiave pubblica dell'utente che vuole accedere alle istanze.

```
ssh-keygen -t rsa -b 2048
nova keypair-add --pub_key ~/.ssh/id_rsa.pub adminkey
```

Listato 15: Abilitazione accesso alle istanze di Nova con chiave pubblica

Fig. 60: Chiave pubblica aggiunta a Nova

```
HeatTemplateFormatVersion: '2012-12-12'
Description: Template for setting up an AutoScaled OpenShift Origin
environment
Parameters:
  KeyName:
    Description: Name of an existing keypair to enable SSH access to the
instances
    Type: String
    MinLength: '1'
    MaxLength: '64'
    AllowedPattern: '[-_ a-zA-Z0-9]*'
  Prefix:
    Description: Your DNS Prefix
    Type: String
    Default: example.com
```

```
  UpstreamDNS:
    Description: Upstream DNS server
    Type: String
    Default: 8.8.8.8
  BrokerServerFlavor:
    Description: Flavor of broker server
    Type: String
    Default: m1.small
    AllowedValues: [m1.small, m1.medium, m1.large, m1.xlarge]
    ConstraintDescription: Must be a valid server flavor
  NodeServerFlavor:
    Description: Flavor of node servers
    Type: String
    Default: m1.small
    AllowedValues: [m1.small, m1.medium, m1.large, m1.xlarge]
    ConstraintDescription: Must be a valid server flavor
  NodeCountMinimum:
    Description: Minimum number of nodes to scale down to
    Type: String
    Default: '1'
    AllowedPattern: '[0-9]*'
  NodeCountMaximum:
    Description: Maximum number of nodes to scale up to
    Type: String
    Default: '3'
    AllowedPattern: '[0-9]*'
Mappings:
  JeosImages:
    Broker:
      Image: F18-x86_64-openshift-origin-broker-cfntools
    Node:
      Image: F18-x86_64-openshift-origin-node-cfntools
Resources:
  OpenshiftUser:
    Type: AWS::IAM::User
  OpenshiftOriginKeys:
    Type: AWS::IAM::AccessKey
    Properties:
      UserName:
        Ref: OpenshiftUser
  OpenshiftOriginNodeGroup:
    Type: AWS::AutoScaling::AutoScalingGroup
    DependsOn: BrokerWaitCondition
    Properties:
      AvailabilityZones: []
      LaunchConfigurationName:
        Ref: NodeLaunchConfig
      MinSize:
        Ref: NodeCountMinimum
      MaxSize:
        Ref: NodeCountMaximum
      LoadBalancerNames: []
  OpenshiftOriginScaleUpPolicy:
    Type: AWS::AutoScaling::ScalingPolicy
    Properties:
      AdjustmentType: ChangeInCapacity
```

```
      AutoScalingGroupName:
        Ref: OpenshiftOriginNodeGroup
      Cooldown: '120'
      ScalingAdjustment: '1'
  OpenshiftOriginScaleDownPolicy:
    Type: AWS::AutoScaling::ScalingPolicy
    Properties:
      AdjustmentType: ChangeInCapacity
      AutoScalingGroupName:
        Ref: OpenshiftOriginNodeGroup
      Cooldown: '60'
      ScalingAdjustment: '-1'
  NodeScaleUp:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmDescription: Scale-up if event received from broker
      MetricName: Heartbeat
      Namespace: system/linux
      Statistic: SampleCount
      Period: '60'
      EvaluationPeriods: '1'
      Threshold: '0'
      AlarmActions: [{Ref: OpenshiftOriginScaleUpPolicy}]
      Dimensions:
      - Name: AutoScalingGroupName
        Value:
          Ref: OpenshiftOriginNodeGroup
      ComparisonOperator: GreaterThanThreshold
  NodeScaleDown:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmDescription: Scale-down if event received from broker
      MetricName: Heartbeat
      Namespace: system/linux
      Statistic: SampleCount
      Period: '60'
      EvaluationPeriods: '1'
      Threshold: '0'
      AlarmActions: [{Ref: OpenshiftOriginScaleDownPolicy}]
      Dimensions:
      - Name: AutoScalingGroupName
        Value:
          Ref: OpenshiftOriginNodeGroup
      ComparisonOperator: GreaterThanThreshold
 OpenShiftOriginSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Standard firewall rules
      SecurityGroupIngress:
      - {IpProtocol: udp, FromPort: '53', ToPort: '53', CidrIp: 0.0.0.0/0}
      - {IpProtocol: tcp, FromPort: '53', ToPort: '53', CidrIp: 0.0.0.0/0}
      - {IpProtocol: tcp, FromPort: '22', ToPort: '22', CidrIp: 0.0.0.0/0}
      - {IpProtocol: tcp, FromPort: '80', ToPort: '80', CidrIp: 0.0.0.0/0}
      - {IpProtocol: tcp, FromPort: '443', ToPort: '443', CidrIp: 0.0.0.0/0}
      - {IpProtocol: tcp, FromPort: '8000', ToPort: '8000', CidrIp:
0.0.0.0/0}
```

```
      - {IpProtocol: tcp, FromPort: '8443', ToPort: '8443', CidrIp:
0.0.0.0/0}
  BrokerWaitHandle:
    Type: AWS::CloudFormation::WaitConditionHandle
  BrokerWaitCondition:
    Type: AWS::CloudFormation::WaitCondition
    DependsOn: BrokerInstance
    Properties:
      Handle:
        Ref: BrokerWaitHandle
      Timeout: '6000'
  BrokerInstance:
    Type: AWS::EC2::Instance
…
 NodeLaunchConfig:
    Type: AWS::AutoScaling::LaunchConfiguration
..
```

Listato 16: Template OpenshiftAutoScaling.yaml

Il Listato 16 mostra una porzione del template *OpenshiftAutoScaling.yaml* reperibile da [77], evidenziando in grassetto le parti salienti. Tale template permette di avere la configurazione di istanze Broker/Nodo mappate in Glance, in modo tale da ottenere un auto scaling di istanze Nodo, agendo da Load Balancer a monte della piattaforma di sviluppo applicazioni, la quale è auto scalante sul versante delle applicazioni a valle (a seconda del numero di Gear e di nodi).

4.3.3.2.1 Flavor

È possibile passare dei parametri ai template di cui viene effettuato il deploy, attraverso il programma heat-cfn, e uno di questi parametri è relativo al tipo di istanza da generare. Nova, come EC2, definisce dei flavor relativi alle istanze, personalizzabili, attraverso i quali vengono dichiarate le quantità di risorse previste per le stesse istanze, rispettivamente in termini di numero di CPU, RAM, disco, volumi e Swap. La Fig. 61 illustra la lista dei flavor utilizzabili, visualizzabili da Horizon nella sezione apposita. Come si può vedere, è stato aggiunto un flavor custom “m1.shift” a scopo dimostrativo.

Flavors

+ Create Flavor

	Flavor Name	VCPUs	RAM	Root Disk	Ephemeral Disk	Swap Disk	ID	Azioni
	m1.tiny	1	512MB	0	0	0MB	1	Edit Flavor More
	m1.shift	1	592MB	15	0	1184MB	cc6fabab-5c1f-40f6-bb1f-c4bf2c770174	Edit Flavor More
	m1.small	1	2048MB	20	0	0MB	2	Edit Flavor More
	m1.medium	2	4096MB	40	0	0MB	3	Edit Flavor More
	m1.large	4	8192MB	80	0	0MB	4	Edit Flavor More
	m1.xlarge	8	16384MB	160	0	0MB	5	Edit Flavor More

Fig. 61: Flavor di Nova

Il template utilizza un flavor m1.small come impostazione di default, ma qualora, come in questo caso, si volesse usare un flavor di dimensioni minori, è possibile aggiungerlo alla lista degli AllowedValues, ad esempio:

```
AllowedValues: [m1.tiny,m1.small, m1.medium, m1.large, m1.xlarge]
```

dopo questa modifica, il valore m1.tiny passato in parametro al template tramite heat, avrà effetto sul deploy e sarà creata un'istanza con 512 MB di RAM.

4.3.3.2.2 Immagini JeOS

Si dichiarano nel template il nome delle istanze da configurare sulla base delle immagini JeOS costruite precedentemente. È possibile definire più istanze sullo stesso template, con o senza dipendenze, che fanno riferimento ad una o più immagini JeOS.

4.3.3.2.3 AWS::AutoScaling::ScalingPolicy

In questa sezione del template, viene determinato il modo in cui viene effettuato lo scaling dell'istanza. La documentazione ufficiale di AWS definisce tre tipi di

comportamenti nello scaling [81]:

1. Scaling per mantenere un numero fisso di istanze;
2. Scaling on demand;
3. Scaling in base ad uno scheduling.

Il caso corrente fa riferimento allo scaling on demand, per il quale si definisce un gruppo di Auto Scaling (AWS::AutoScaling::AutoScalingGroup) per scalare automaticamente in base a determinate condizioni, che si possono dichiarare nei template. Un gruppo di Auto Scaling usa una combinazione di policy e di Alarm per determinare quando si verifichino delle condizioni per effettuare lo scaling. Il concetto di Alarm, definito in CloudWatch, si riferisce ad un oggetto che effettua il monitoraggio su una singola metrica, definita sempre nel template, per un certo lasso di tempo. Tale metrica, può essere ad esempio la percentuale media di utilizzo della CPU delle istanze definite nel gruppo di Auto Scaling.

La policy di auto scaling è definita su un gruppo di nodi, la cui cardinalità può essere definita come parametro in ingresso al template, ed è vincolata alla Wait Condition sull'istanza del nodo Broker. Tale comportamento, vincola il processo di auto scaling (e tutta l'esecuzione del template) all'avvenuta creazione del nodo Broker, componente essenziale della platform. I parametri che definiscono lo scaling sono `Cooldown` e `ScalingAdjustment` i quali definiscono rispettivamente il tempo di attesa previsto per lo scaling, ed il numero di istanze da scalare. In questo caso, viene creata una sola nuova istanza con tempo di attesa di 120 secondi per i nuovi eventi e/o richieste di istanze.

4.3.3.2.4 AWS::CloudWatch::Alarm

La metrica di auto scaling risiede nel Broker, il quale attraverso CloudWatch, definisce la metrica che genera l'evento di scaling dei nodi. Nella configurazione del Broker attraverso, vengono definiti nel template due script, notify_scale_up e

notify_scale_down i quali eseguono una metrica Heartbeat [82], per un High Availability di istanza, a monte di quello di servizio offerto da HAProxy nei Gear principale e nel web_proxy del Broker di Openshift. La logica risiede nel Broker e le decisioni sullo scaling da effettuare vengono fatte attraverso NodeManager, un framework per automatizzare gli eventi di scaling dei nodi, presente sullo stesso Broker. Le informazioni prese per effettuare una scelta in merito a quando e cosa scalare vengo raccolte dall'architettura di Openshift, la quale in questa scenario, è integrata in un sistema di infrastruttura al quale può richiedere le risorse necessarie, on-demand.

4.3.3.2.5 AWS::CloudFormation::WaitConditionHandle

Nei template compatibili con CloudFormation è possibile definire delle dipendenze fra le risorse descritte. Nel caso specifico, è stata espressa una dipendenza dell'istanza Nodo rispetto al Broker, che è l'entità principale all'interno di Openshift. Heat-cfn rimane in attesa del signaling dal Broker, e successivamente viene lanciata l'istanza del nodo. Se il Broker non completa le sue procedura, il sistema rimarrà sempre in attesa del Broker fino allo scadere di un timeout che sancisce il fallimento del deploy del template.

4.3.3.2.6 Deploy

È possibile effettuare il deploy di due template presenti in [77], uno con una semplice configurazione Broker/Nodo senza HA di istanza *OpenShift.template*, e l'altro come quello appena descritto *OpenShiftAutoScaling.yaml*. In entrambi i casi, il deploy avviene attraverso il programma heat-cfn come segue:

```
heat-cfn create openshift --template-file=OpenShift.template
-parameters="KeyName=adminkey;Prefix=gridlabunical.net;UpstreamDNS=160.97.5.13"
```

Listato 17: Deploy di Openshift con Heat

Successivamente viene eseguito il template per l'orchestrazione delle risorse descritte e vengono lanciate le due istanze. È possibile verificare l'avvenuto conseguimento del deploy con lo stesso comando:

```
heat-cfn list

<ListStacksResponse>
  <ListStacksResult>
    <StackSummaries>
      <member>
<StackId>arn:openstack:heat::3408d4f1670648ec895c8e3d7b1cc4c7:stacks/openshift/92f3
29f4-bc32-4dfa-8c65-1d2685e0c8a8</StackId>
        <LastUpdatedTime>2013-05-31T13:43:46Z</LastUpdatedTime>
        <TemplateDescription>Template for setting up an OpenShift Origin
environment</TemplateDescription>
        <StackStatusReason>Stack successfully created</StackStatusReason>
        <CreationTime>2013-05-31T13:42:17Z</CreationTime>
        <StackName>openshift</StackName>
        <StackStatus>CREATE_COMPLETE</StackStatus>
      </member>
    </StackSummaries>
  </ListStacksResult>
</ListStacksResponse>
```

Listato 18: Deploy avvenuto con successo

Istanze

	Progetto	Host	Nome	Indirizzo IP	Dimensione	Stato	Task	Stato alimentazione
	admin	moon4.deis.unical.it	openshift.NodeInstance	192.168.32.3	m1.tiny | 512MB RAM | 1 VCPU | 0 Disk	Active	None	Running
	admin	moon4.deis.unical.it	openshift.BrokerInstance	192.168.32.2	m1.tiny | 512MB RAM | 1 VCPU | 0 Disk	Active	None	Running

Displaying 2 items

Fig. 62: Istanze in esecuzione

La Fig.62 illustra il deploy di esempio mostrato nel Listato 17. Broker e Nodo sono eseguite come due istanze separate, con le configurazioni passate nel Listato 17, ovvero con l'abilitazione della flavor m1.tiny, e la descrizione delle risorse nel template.

La configurazione illustrata è solo dimostrativa, poiché per un deploy funzionante è necessario avere a disposizione almeno 8 GB di RAM sulla macchina host, ed assegnare a ciascun istanza almeno 1 GB di RAM, o una flavor m1.small (2GB). Le istanze sono in esecuzione e accessibili tramite SSH ad un indirizzo IP generato, a partire dalla maschera definita nel file di configurazione di Nova, come illustrato nel Listato 19:

```
ps aux | grep qemu

qemu     17049  3.4  7.0 1596624 135408 ?     Sl   22:58   0:13 /usr/libexec/qemu-kvm
-name instance-00000012 -S -M rhel6.4.0 -cpu core2duo,+lahf_lm,+dca,+pdcm,+xtpr,
+cx16,+tm2,+est,+vmx,+ds_cpl,+dtes64,+pbe,+tm,+ht,+ss,+acpi,+ds -enable-kvm -m 512
-smp 1,sockets=1,cores=1,threads=1 -uuid c5e2662e-b0ce-4ede-875b-94640724a667
-smbios type=1,manufacturer=Red Hat Inc.,product=OpenStack Nova,version=2013.1.1-
2.el6,serial=34313635-3636-435a-4a37-303730304c4a,uuid=c5e2662e-b0ce-4ede-875b-
94640724a667 -nodefconfig -nodefaults -chardev
socket,id=charmonitor,path=/var/lib/libvirt/qemu/instance-00000012.monitor,server,nowait
-mon chardev=charmonitor,id=monitor,mode=control -rtc base=utc,driftfix=slew -no-kvm-
pit-reinjection -no-shutdown -device piix3-usb-uhci,id=usb,bus=pci.0,addr=0x1.0x2 -drive
file=/var/lib/nova/instances/c5e2662e-b0ce-4ede-875b-
94640724a667/disk,if=none,id=drive-virtio-disk0,format=qcow2,cache=none -device virtio-
```

```
blk-pci,scsi=off,bus=pci.0,addr=0x4,drive=drive-virtio-disk0,id=virtio-disk0,bootindex=1
-netdev tap,fd=22,id=hostnet0,vhost=on,vhostfd=23 -device virtio-net-
pci,netdev=hostnet0,id=net0,mac=fa:16:3e:fb:8a:96,bus=pci.0,addr=0x3 -chardev
file,id=charserial0,path=/var/lib/nova/instances/c5e2662e-b0ce-4ede-875b-
94640724a667/console.log -device isa-serial,chardev=charserial0,id=serial0 -chardev
pty,id=charserial1 -device isa-serial,chardev=charserial1,id=serial1 -device usb-
tablet,id=input0 -vnc 160.97.24.114:0 -k en-us -vga cirrus -device virtio-balloon-
pci,id=balloon0,bus=pci.0,addr=0x5

qemu     17757  7.5 10.2 1303344 196940 ?      Sl   22:59   0:24 /usr/libexec/qemu-kvm
-name instance-00000013 -S -M rhel6.4.0 -cpu core2duo,+lahf_lm,+dca,+pdcm,+xtpr,
+cx16,+tm2,+est,+vmx,+ds_cpl,+dtes64,+pbe,+tm,+ht,+ss,+acpi,+ds -enable-kvm -m 512
-smp 1,sockets=1,cores=1,threads=1 -uuid b7c37f0c-1db1-4281-b096-5f2d98669f97
-smbios type=1,manufacturer=Red Hat Inc.,product=OpenStack Nova,version=2013.1.1-
2.el6,serial=34313635-3636-435a-4a37-303730304c4a,uuid=b7c37f0c-1db1-4281-b096-
5f2d98669f97 -nodefconfig -nodefaults -chardev
socket,id=charmonitor,path=/var/lib/libvirt/qemu/instance-00000013.monitor,server,nowait
-mon chardev=charmonitor,id=monitor,mode=control -rtc base=utc,driftfix=slew -no-kvm-
pit-reinjection -no-shutdown -device piix3-usb-uhci,id=usb,bus=pci.0,addr=0x1.0x2 -drive
file=/var/lib/nova/instances/b7c37f0c-1db1-4281-b096-5f2d98669f97/disk,if=none,id=drive-
virtio-disk0,format=qcow2,cache=none -device virtio-blk-
pci,scsi=off,bus=pci.0,addr=0x4,drive=drive-virtio-disk0,id=virtio-disk0,bootindex=1 -netdev
tap,fd=22,id=hostnet0,vhost=on,vhostfd=24 -device virtio-net-
pci,netdev=hostnet0,id=net0,mac=fa:16:3e:c1:2d:de,bus=pci.0,addr=0x3 -chardev
file,id=charserial0,path=/var/lib/nova/instances/b7c37f0c-1db1-4281-b096-
5f2d98669f97/console.log -device isa-serial,chardev=charserial0,id=serial0 -chardev
pty,id=charserial1 -device isa-serial,chardev=charserial1,id=serial1 -device usb-
tablet,id=input0 -vnc 160.97.24.114:1 -k en-us -vga cirrus -device virtio-balloon-
pci,id=balloon0,bus=pci.0,addr=0x5
```

Listato 19: Istanze in esecuzione su KVM

```
ssh ec2-user@192.168.32.3
Last login: Fri May 31 09:46:45 2013 from 192.168.32.1
[ec2-user@openshift ~]$
```

Listato 20: Broker e Nodo accessibili via SSH

```
yum install -y lynx
lynx https://192.168.32.3/console
```

Listato 21: Accesso alla Console di Openshift da CLI

Il Listato 21 mostra come accedere alla Console di Openshift in maniera testuale all'interno della macchina host dove risiede Openstack; qualora si voglia effettuare l'accesso tramite browser, è necessario impostare una configurazione DNS apposita oppure realizzare un semplice port forwarding tramite iptables come segue:

```
#!/bin/sh

iptables -A PREROUTING -t nat -i eth0 -p tcp --dport 9180 -j DNAT --to 192.168.32.2:80
iptables -A FORWARD -p tcp -d 192.168.32.2 --dport 9180 -j ACCEPT
iptables -A INPUT -p tcp --dport 9180 -j ACCEPT
iptables -A OUTPUT -p tcp --sport 9180 -j ACCEPT
iptables -A FORWARD -p tcp --sport 9180 --dport 9180 -j ACCEPT

iptables -A PREROUTING -t nat -i eth0 -p tcp --dport 9181 -j DNAT --to 192.168.32.2:443
iptables -A FORWARD -p tcp -d 192.168.32.2 --dport 9181 -j ACCEPT
iptables -A INPUT -p tcp --dport 9181 -j ACCEPT
iptables -A OUTPUT -p tcp --sport 9181 -j ACCEPT
iptables -A FORWARD -p tcp --sport 9181 --dport 9181 -j ACCEPT
```

Listato 22: Port forwarding 80->9180, 443->9181

Il Listato 22 illustra un semplice script Bash che permette di mappare la porta 443 del Broker su una porta della macchina host dove risiede Openstack, in questo caso 9181. Questo perché sulle porta 80 della macchina è in esecuzione Horizon su base Apache, e in caso in cui non si configurino appositamente i DNS per rendere il Broker accessibile dall'esterno con delle entry apposite, insieme a delle regole di iptables, le istanze create non sono accessibili dall'esterno. La Fig. 63 mostra la Console Web di Openshift.
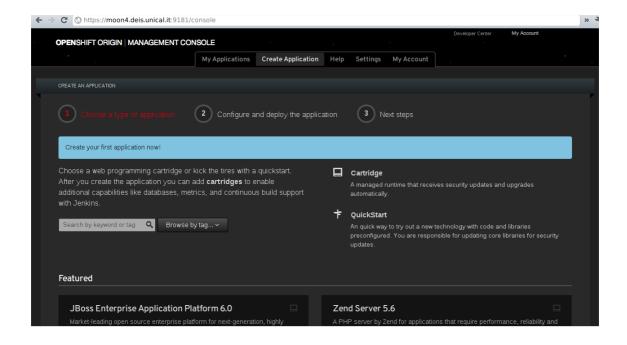
Fig. 63: Console Web di Openshift

Il deploy delle applicazioni avviene attraverso la CLI, ed è possibile creare delle applicazioni scalabili dichiarandolo espressamente durante la creazione di nuove applicazioni con il programma rhc, ad esempio per un'applicazione web su base JavaEE automaticamente scalabile fino ad un massimo di 3 Gear:

```
rhc create-app Esempio jbossas7 -s
rhc scale-cartridge jbossas7 -a Esempio --min 1 --max 3
```

Listato 23: Creazione di una Web App Enterprise auto-scalabile

Accessibile successivamente all'indirizzo contenente il namespace ed il nome dell'applicazione, ad esempio: esempio-gridlab.gridlabunical.net

4.3.4 Contributi

https://bugzilla.redhat.com/show_bug.cgi?id=969066

https://review.openstack.org/31073

https://review.openstack.org/#/c/30629/

http://openstack.redhat.com/Deploy_Heat_and_launch_your_first_Application

https://github.com/blues-man

CONCLUSIONI

Io t'ho dato le ali. Volteggerai, librato,
agevolmente sulla terra e il mare.
..ti faranno scorta i doni fulgidi
delle Muse dai serti di viole.
Canto sarai nei secoli, per chi del canto è vago,
Finché la terra duri e duri il sole. [85]

L'infrastruttura proposta spazia fra i modelli di servizio all'interno del paradigma del cloud computing, offrendo un modello di erogazione di servizi web, che si basa sulla combinazione di tecniche e soluzioni consolidate. Il cloud ibrido è la soluzione ideale per le organizzazioni che hanno necessità di usufruire di risorse interne ed esterne, distribuendo il carico di lavoro nella maniera più opportuna, ed il cloud stesso non è soltanto uno strumento esterno da contabilizzare e da integrare nel prezzo della produzione interna, ma il modello promotore e fautore dell'intera offerta di IT.

Openstack rappresenta la scelta ideale soluzioni di cloud privato e ibrido, favorendo sia in contesti aziendali, che di attività di ricerca, un metodo di organizzazione e razionalizzazione delle risorse fisiche, offerte sul modello delle Service Oriented Archirecture.

Openshift è un progetto di nuova concezione di cloud ibrido di piattaforma, che fornisce un ambiente dinamico ed estendibile per lo sviluppo di applicazioni cloud-ready. Nell'elaborato, viene mostrato come sia possibile integrare una piattaforma di sviluppo enterprise, basata su modello PaaS, con un'infrastruttura predisposta allo scaling e alla distribuzione di risorse generalmente virtualizzate, basata su modello IaaS.

L'interconnessione di livello, effettuata tramite Openstack ed Openshift, rappresenta una soluzione innovativa attraverso la quale è possibile far convergere i benefici derivanti da entrambi i modelli utilizzati, verso un'unica completa soluzione di computazione on-demand, che si adatta sia alle esigenze di integrità e disponibilità dei servizi, piuttosto che alla portabilità del codice e all'estensione dei modelli di deployment.

Si è visto come la progettazione di un'infrastruttura multi-servizio di multi-livello possa risolvere il problema della disponibilità dei servizi stessi, con un doppio sistema di High Availability di istanza e di servizio, che possa mantenere costante il throughput di risposte alle richieste effettuate alle applicazioni nel cloud. L'orchestrazione delle risorse in Openstack, attraverso Heat, permette di descrivere le componenti sensibili al guasto e di descrivere delle politiche di *fault tolerance* grazie ad un sistema di templating pienamente compatibile con AWS, pertanto nella direzione della portabilità nelle diverse soluzioni di cloud, le quali tendono a convergere verso una maggiore astrazione di erogazione dei servizi per mezzo di apposite API. Tutti i servizi di Openstack espongono delle API RESTful, utilizzate dalle applicazioni nel cloud per la gestire il loro ciclo di vita all'interno del cloud stesso e per la comunicazione fra i servizi per la configurazione dell'ambiente di infrastruttura. La computazione on-demand avviene per mezzo delle semplici richieste HTTP, attraverso delle quali possono essere mandati in produzione diversi ambienti di esecuzione isolati, tuttavia interconnessi secondo la logica del cloud.

Come si è visto, un'applicazione Web Enterprise può essere offerta in una soluzione di cloud ibrido; attraverso una configurazione di Openshift Origin, è possibile eseguire le proprie applicazioni sulla propria infrastruttura, oppure distribuire il carico di risorse su più data center, ed offrire le stesse applicazioni attraverso Internet in maniera del tutto trasparente per l'utente finale e per lo sviluppatore, al quale è fornita anche la scelta sul tipo di ambiente sul quale progettare le proprie applicazioni. Lo scopo di questo elaborato è in effetti mostrare come, anche attraverso il cloud, sia possibile anche effettuare una scelta, che permetta di divenire non più soltanto utilizzatori finali di servizi, ma attori principali del modello in cui sono offerti tali servizi e sulla scelta degli stessi in base alle proprie necessità. È stato mostrato come un'applicazione JavaEE possa essere offerta sul cloud in maniera semplice attraverso uno git push, e come la stessa possa essere pienamente personalizzabile con scelta di runtime prefabbricati o DoItYourself (DIY). Con l'avvento costante di nuove tecnologie di sviluppo, a partire da HTML5, Javascript MVC, ma anche ad esempio Erlang o Go, risulta strategico ottenere un ambiente di sviluppo dinamico ed altamente estendibile, sempre nella logica del cloud.

Il concetto di Cartridge personalizzato rende qualsiasi tecnologie potenzialmente disponibile, progettata a priori per essere auto-scalabile, distribuita su larga scala ed interconnessa su più cloud. Openshift implementa High Availability di servizio grazie al Broker, nodo coordinatore della piattaforma, il quale attraverso un HAProxy riesce a scalare automaticamente i contenitori di Cartridge, denominati Gear, su più nodi all'occorrenza. A monte di tutto ciò, è stato mostrato come lo stesso scaling dei nodi possa essere orchestrato con un HA di istanza per mezzo delle API compatibili di CloudFormation e CloudWatch, le quali descrivono la risorsa Broker come oggetto di monitoraggio ed aggiungono all'istanza stessa un meccanismo di Alarm di tipo Heartbeat, il quale nell'esempio mostrato scala automaticamente un nodo in caso di segnalazione da parte del Broker. In questa maniera, il numero di nodi della piattaforma diverrebbe dinamico, e qualora essa debba essere offerta su provider di cloud pubblico, verrebbe tariffata la computazione dei nodi aggiuntivi solo se realmente necessari, secondo il consueto concetto di economia di scala del cloud. Ma è anche un metodo di razionalizzazione di risorse, virtualizzate o “bare-metal” che siano, certamente condivise, pertanto poter razionalizzare e distribuire in modo armonico la quantità di computazione, risulta strategico nella logica del cloud dove il leitmotiv è l'ottimizzazione.

Le soluzioni future che potrebbero essere implementate sono molteplici. La proposta in questione può essere un punto di partenza per il raggiungimento di più obiettivi su più fronti.

Al momento, nell'architettura di Openshift, il coordinamento risiede sul nodo Broker, che mantiene la comunicazione con i nodi contenenti tutti i Gear usufruibile dalle applicazioni. L’intero traffico HTTP, indirizzato verso le applicazioni, viene inoltrato verso il nodo contenente il Gear primario dell'applicazione, per mezzo di HAProxy verso altri Gear ed altri Nodi coinvolti. Si dovrebbe pertanto prevedere un meccanismo che permetta alle applicazioni di sopravvivere anche in caso di guasti ad HAProxy, coinvolgendo l'interazione dell'infrastruttura sottostante. Come discusso nei precedenti capitoli, le applicazioni web in Openshift sono mappate da una tripla entry DNS, Load Balancer di Gear, e un sistema di backend del Gear (Cartridge, Stickshift API). L'insieme di queste componenti forma la tabella di routing di Openshift, la quale viene

mantenuta dal Broker per tutti gli endpoint, non soltanto quelli relativi al protocollo HTTP (quindi connessioni TCP, UDP, ICMP interne all'architettura).

Pertanto si potrebbe realizzare un sistema di notifiche esterne all'architettura, che sia pienamente integrato con l'infrastruttura sottostante e dunque, volendo continuare con questo modello, ad esempio realizzare l'integrazione delle API di Openstack di Neutron per la tabella di routing, progettando un meccanismo di HA di istanza (o di sistema), sulla base di quello mostrato in questo elaborato. Questa soluzione va anche nella direzione degli sforzi della community, la quale sostiene la progettazione e lo sviluppo della piattaforma.

Altre implementazioni future possibili per la realizzazione di un'infrastruttura che supporti lo sviluppo Web Enterprise, potrebbero considerare sempre Openstack, tuttavia mediante riguardare l'utilizzo di Juju. Come accennato, il servizio di orchestrazione su base Ubuntu Linux, si predispone ad una configurazione semplice di servizi e di High Availabity di servizio, mediante uno scaling delle risorse in maniera semplice e dichiarativo.

Attraverso questa soluzione, avrebbe senso la realizzazione di una piattaforma di sviluppo Web Enterprise con Java su base Terracotta, disponibile come Charm autoscalante all'interno del servizio, potendo usufruire di un sistema di clusterizzazione automatica della JVM basato su BigMemory, sul cloud privato e/o interconnesso con il cloud pubblico di Amazon.

Inoltre, si potrebbe prevedere l'utilizzo di CloudFoundry, orchestrato sempre tramite Juju su base Openstack, magari con ESX.

Come si evince dalle argomentazioni trattate in questa tesi, è possibile muoversi su più direzioni, valutando tutte le opzioni offerte dai vari progetti. In questo elaborato è stata effettuata una scelta, che si è rivelata promettente e che va nella direzione dove si incentra la maggior parte degli sforzi all'interno del cloud computing, ovvero quella dell'interoperabilità e dell'indipendenza dai provider. È sempre più crescente il bisogno di astrazione e di interconnessioni fra i cloud, ed Openstack, insieme ad Openshift, permette di ottenere una soluzione inter-cloud open source altamente scalabile e distribuita, seguita da una larga community di sviluppatori e di realtà aziendali che

convergono sullo stesso obiettivo: rendere il cloud alla portata di tutti. Free as in Freedom.

RINGRAZIAMENTI

Ringrazio innanzitutto il mio relatore, il Prof. Talia, per la disponibilità, il supporto didattico e la fiducia nelle scelte, certamente non scontate, intraprese per realizzare questo elaborato, su tecnologie ed approcci del tutto nuovi nel cloud computing. Vorrei anche ringraziare le persone del dipartimento che mi hanno seguito durante la stesura della tesi nel laboratorio di Grid computing, in particolare l'Ing. Trunfio e l'Ing. Lackovic, sempre disponili, a cui ho fatto spesso riferimento per suggerimenti e spunti.

Ringrazio Fabrizio, promotore di questa iniziativa che mi ha assistito nella ricerca delle soluzioni più opportune e coerenti con gli obiettivi prefissati. Ringrazio inoltre Marco e Tommaso, che insieme a Fabrizio hanno fondato Hictech, l'azienda dove sono cresciuto professionalmente e che mi ha accompagnato in questi anni di studio. Grazie anche a Tanaza, “cloud inside”, in particolare a Matteo, Rocco, Danabel, Cristian e Sebastiano.

Ringrazio tutto il Telelab, il mio “nono piano” di MITtiana memoria, luogo di studio, di svago e di radiazioni gratuite. Ringrazio in particolare Peppino, Floriano, e Mauro per il supporto e la disponibilità, e Andrea, Strangis, Francesco, Antofrage, i Gracchi, Ciro e gli storici Loris e Antonio per aver reso meno tragiche, tremende infinite giornate di studio. Volevo inoltre ringraziare tutti gli amici che mi hanno accompagnato in questo lungo viaggio, a cominciare da Vincenzo Palermo, sempre presente, e Maria, che non finirò mai di ringraziare anche per la seconda laurea, e poi Fabrizia, CarVela, Simona e i “soviet” della P2 ovvero i “pazzi” Peppone, Bicicletta, Tarlo, i “nerd” @ono-sendai, garulf, ciccillo, lethalman, spike e i “blues” Andrea Angol e Lorenzo Blues. Che Blues!

Ringrazio la mia famiglia, mia madre, mio padre, mio fratello e mia nonna, che mi hanno sempre sostenuto in questo lungo cammino, e alla fine, ce l'abbiamo fatta!

Infine, ringrazio Alessia, che mi ha aiutato a concludere questo percorso, che non ritenevo più alla mia portata. Averti accanto in un tragitto così tanto difficile, mi ha dato la forza di continuare e di crederci, con la sicurezza di cui avevo bisogno. Anche questo risultato, è grazie a te.

BIBLIOGRAFIA

[1] Omaggio a Cesare, v. incipit De Bello Gallico, Liber I

[2] R.Giordanelli, C.Mastroianni - The Cloud Computing Paradigm: Characteristics, Opportunities and Research Issues, ICAR-CNR

[3] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In Grid Computing Environments Workshop, 2008. GCE '08, pp 1–10.

[4] P.Mell, T. Grance - The NIST Definition of Cloud Computing. NIST Special Publication 800-145, pp 2-4

[5] Cloud Computing White Paper - IBM Global Technology Services, pp. 4 -5

[6] M. Sheehan. Cloud computing expo: Introducing the cloud pyramid. SYS-CON Media, 2008. http:// cloudcomputing.sys-con.com/node/609938

[7] K. Stanoevska-Slabeva, T. Wozniak, S. Ristol – Grid and Cloud Computing, A Business Perspective on Technology and Applications, Springer, 2012, pp. 52-54

[8] Goldberg, R. Architectural Principles for Virtual Computer Systems. PhD thesis, National Technical Information Service, February 1973.

[9] F. Van Der Molen - Get Ready for Cloud Computing: A Comprehensive Guide to Virtualization and Cloud Computing, Van Haren Publishing, 2010, pp. 29-31

[10] VMware White Paper - Understanding Full Virtualization, Paravirtualization and Hardware Assist, 2007

http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf

[11] C. Chaubal - The Architecture of VMware ESXi , VMware White Paper, 2013

http://www.vmware.com/files/pdf/ESXi_architecture.pdf

[12] MSDN Library – Hyper-V Architecture

http://msdn.microsoft.com/en-US/library/cc768520(v=bts.10).aspx

[13] Y. Goto - Kernel-based Virtual Machine Technology , FUJITSU Sci. Tech. J., Vol. 47, No. 3 (July 2011)

[14] Libvirt Docs, Descrizione obiettivi

http://libvirt.org/goals.html

[15] Z.Mahmood, R. Hill - Cloud Computing for Enterprise Architectures, Springer, 2011, pp.98-101

[16] J.Brodkin – Gartner: Seven cloud-computing security risks, http://www.networkworld.com/news/2008/070208-cloud.html, 2008.

[17] Top Threats to Cloud Computing, Survey Results Update 2012, Cloud Security Alliance,
https://downloads.cloudsecurityalliance.org/initiatives/top_threats/Top_Threats_Cloud_Computing_Survey_2012.pdf

[18] B.Grobauer, T. Walloschek, E. Stöcker - Understanding Cloud Computing Vulnerabilities, Siemens, 2012

[19] J. Mc Carthy, da un discorso nel MIT Centennial del 1961 - Architects of the Information Society, Thirty-Five Years of the Laboratory for Computer Science at MIT, Hal Abelson

[20] S.Levy – Hackers - Gli eroi della rivoluzione informatica, Edizioni Shake, 2008, pp. 34-42

[21] B.P.Rimal, E.Choi, I. Lamb – A Taxonomy and Survey of Cloud Computing Systems, 2009 Fifth International Joint Conference on INC, IMS, and IDC, IEEE Computer Society

[22] Adattata da

http://sajojacob.com/blog/wp-content/uploads/2010/10/ServiceModel_thumb.jpg

[23] AWS Documentation – What is Amazon Web Services?
http://docs.aws.amazon.com/gettingstarted/latest/awsgsg-intro/intro.html

[24] J. Varia, S.Mathew – Overview of Amazon Web Services whitepaper, 2013

http://docs.aws.amazon.com/gettingstarted/

[25] AWS Documentation, AMI

https://aws.amazon.com/amis

[26] Adattata da http://www.janakiramm.net/blog/overview-of-amazon-web-services

[27] AWS Documentation – Getting Started with S3

http://docs.aws.amazon.com/AmazonS3/latest/gsg/GetStartedWithS3.html

[28] Eucalyptus Cloud Architecture

http://www.eucalyptus.com/eucalyptus-cloud/iaas/architecture

[29] http://www.infoworld.com/slideshow/80986/infoworlds-2013-technology-of-the-year-award-winners-210419?source=fssr#slide15

[30] Joyent Developers Resources - A Framework for Selecting the Right Cloud Infrastructure Provider

http://www.joyent.com/content/06-developers/01-resources/43-a-framework-for-selecting-the-right-cloud-infrastructure-provider/121204-aws-whitepaper.pdf

[31] Wikipedia, ZFS

http://en.wikipedia.org/wiki/ZFS

[32] Joyent White Paper - The Joyent Smart Technologies Architecture for Cloud Computing

http://www.joyent.com/content/06-developers/01-resources/06-the-joyent-smart-technologies-architecture-for-cloud-computing/smart-architecture-cloud-computing.pdf

[33] SmartOS Documentation - Why SmartOS: ZFS, KVM, DTrace, Zones and More

http://wiki.smartos.org/display/DOC/Why+SmartOS+-+ZFS%2C+KVM%2C+DTrace%2C+Zones+and+More

[34] OpenNebula Documentation

http://opennebula.org/documentation:rel4.0:intro

[35] Open Cloud Computing Interface,

http://occi-wg.org/

[36] VMware Documentation - Architecting VMware vCloud

http://www.vmware.com/files/pdf/vcat/Architecting-VMware-vCloud.pdf

[37] Windows Azure Documentation – Windows Azure Execution Models

http://www.windowsazure.com/en-us/manage/windows/fundamentals/compute/

[38] Azure Virtual Machines Catalog

http://vmdepot.msopentech.com/List/Index

[39] Windows Azure Documentation – Manage the Availability of Virtual Machines

http://www.windowsazure.com/en-us/manage/windows/common-tasks/manage-vm-availability/

[40] Oxford Economics: Unlocking the Cloud - A Cloud Platform Overview

http://scn.sap.com/docs/DOC-40516

[41] Developers Docs - What is Google App Engine

https://developers.google.com/appengine/docs/whatisgoogleappengine

[42] K. Karan, S. Surendran - BigTable, Dynamo & Cassandra – A Review , International Journal of Electronics and Computer Science Engineering , Volume 2 Number 1, pp. 136-138

[43] T.Chandra, R. Griesemer, J. Redstone - Paxos Made Live, An Engineering Perspective , Google Research, 2006

http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/archive/paxos_made_live.pdf

[44] App Engine Documentation, Datastore Query

https://developers.google.com/appengine/docs/java/datastore/queries#Java_Ancestor_queries

[45] Heroku Documentation, HTTP Routing

https://devcenter.heroku.com/articles/http-routing

[46] Heroku Documentation, Dynos

https://devcenter.heroku.com/articles/dynos

[47] The 12 Factor,

http://www.12factor.net/

[48] CloudFoundry Docs. Architecture

http://docs.cloudfoundry.com/docs/running/architecture/

[49] CloudFoundry Stacks,

https://github.com/cloudfoundry/stacks

[50] CloudFoundry Documentation, BOSH

http://docs.cloudfoundry.com/docs/running/bosh/components/

[51] I. Calvino – Le lezione americane. Sei proposte per il nuovo millennio , La Leggerezza, Mondadori, 2000, 14 ed.

[52] S. Williams – Free as in Freedom (2.0): Richard Stallman and the Free Software Revolution, Free Software Foundation, 2010

http://static.fsf.org/nosvn/faif-2.0.pdf

[53] Openstack Wiki, An Overview

https://wiki.openstack.org/wiki/Main_Page

[54] ARM Virtual Extenstions,

http://www.arm.com/products/processors/technologies/virtualization-extensions.php

[55] Openstack Manuals,

https://github.com/openstack/openstack-manuals

[56] OASIS Topology and Orchestration Specification for Cloud Applications

https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

[57] Openstack Wiki, Heat Orchestration

https://wiki.openstack.org/wiki/Heat

[58] Slidedshare - Exploring Heat,

http://www.slideshare.net/dbelova/openstack-heat-slides

[59] Codice sorgente Heat,

https://github.com/openstack/heat

[60] Puppet Docs, What is Puppet

https://puppetlabs.com/puppet/what-is-puppet/

[61] Juju Docs,

https://juju.ubuntu.com/docs/

[62] TripleO Presentation,

http://www.slideshare.net/hpcloud/deploying-and-managing-openstack-with-heat/20

[63] Openshift Public PaaS,

https://www.openshift.com

[64] Codice sorgente Openshift,

https://github.com/openshift

[65] Openshift Wiki – Architecture Overview

https://www.openshift.com/wiki/architecture-overview

[66] Openshift FAQ – Gears size,

https://www.openshift.com/faq/are-there-different-gear-sizes-and-how-much-do-they-cost

[67] Sviluppo ad integrazione continua con Jenkins,

http://jenkins-ci.org/

[68] Documentazione Cartridge,

http://openshift.github.io/origin/node/file.README.writing_cartridges.html

[69] Little Richard - Tutti Frutti - Here's Little Richard, Speciality Records, 1955

[70] Y. Fain, V. Rasputnis, A. Tartakovsky, V. Gamov – Enterprise Web Development, O'Really, 2013, Introduction

http://enterprisewebbook.com/

[71] JBoss Docs, Getting Started Guide,

https://docs.jboss.org/author/display/AS7/Getting+Started+Guide

[72] JBoss Docs, High Availability Guide,

https://docs.jboss.org/author/display/AS7/High+Availability+Guide

[73] CentOS 6.4 ISO Garr mirror,

http://mi.mirror.garr.it/mirrors/CentOS/6.4/isos/x86_64/

[74] Bug di Anaconda, installazione su penna USB

https://bugzilla.redhat.com/show_bug.cgi?id=568343

[75] Openstack Docs, Heat on Fedora

http://docs.openstack.org/developer/heat/getting_started/on_fedora.html

[76] RDO Wiki, Deploy Heat

http://openstack.redhat.com/Deploy_Heat_and_launch_your_first_Application

[77] Codice sorgente template Heat

https://github.com/openstack/heat-templates

[78] Oz, tool per creazione istanze

https://github.com/clalancette/oz

[79] Docs CloudInit

https://help.ubuntu.com/community/CloudInit

[80] Vagrant, sistema di creazione immagini virtuali

http://www.vagrantup.com/

[81] AWS CloudFormation Docs, ScalingGroup

http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-as-policy.html

[82] AWS CloudWatch Docs, Metriche

http://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/API_PutMetricAlarm.html

[83] Wiki Openstack, Using HA

https://wiki.openstack.org/wiki/Heat/Using-HA

[84] DeltaCloud, API di astrazione dei servizi nel cloud

http://deltacloud.apache.org/index.html

[85] Teognide, Frammenti, vv. 237-238, 250-252